When quality_id is greater than 0 and setOfRefLayerSlices contains more than one slice, the following constraints shall be obeyed.

a) When slice_type specifies an EP or EB slice, the value of the syntax element num_ref_idx_l0_active_minus1 (either the value transmitted in the slice header when num_ref_idx_active_override_flag is equal to 1 or the value of the referenced picture parameter set when num_ref_idx_active_override_flag is equal to 0) shall be the same across all slices of the set setOfRefLayerSlices.

b) When slice_type specifies an EB slice, the value of the syntax element num_ref_idx_l1_active_minus1 (either the value transmitted in the slice header when num_ref_idx_active_override_flag is equal to 1 or the value of the referenced picture parameter set when num_ref_idx_active_override_flag is equal to 0) shall be the same across all slices of the set setOfRefLayerSlices.

c) All elements of the syntax structure ref_pic_list_reordering( ) shall be the same across all slices of the set setOfRefLayerSlices.

d) When slice_type specifies an EP slice, the syntax element weighted_pred_flag shall be the same across all slices of the set setOfRefLayerSlices and the following applies.

   i. When weighted_pred_flag is equal to 1 and no_inter_layer_pred_flag is equal to 0 for any of the slices of the set setRefLayerSlices, base_pred_weight_table_flag shall be the same across all slices of the set setOfRefLayerSlices.

   ii. When weighted_pred_flag is equal to 1 for any of the slices of the set setRefLayerSlices and pred_weight_table( ) is present in any of the slices of the set setOfRefLayerSlices, pred_weight_table( ) shall be present in all slices of the set setOfRefLayerSlices and all syntax elements inside the syntax structure pred_weight_table( ) shall be the same across all slices of the set setOfRefLayerSlices.

e) When slice_type specifies an EB slice, the syntax element weighted_bipred_idc shall be the same across all slices of the set setOfRefLayerSlices and the following applies.

   i. When weighted_bipred_idc is equal to 1 and no_inter_layer_pred_flag is equal to 0 for any of the slices of the set setRefLayerSlices, base_pred_weight_table_flag shall the same across all slices of the set setOfRefLayerSlices.

   ii. When weighted_bipred_idc is equal to 1 for any of the slices of the set setRefLayerSlices and pred_weight_table( ) is present in any of the slices of the set setOfRefLayerSlices, pred_weight_table( ) shall be present in all slices of the set setOfRefLayerSlices and all syntax elements inside the syntax structure pred_weight_table( ) shall be the same across all slices of the set setOfRefLayerSlices.

When quality_id is greater than 0, the following syntax elements (that are not present) shall be inferred to be equal to the corresponding syntax elements of the slice headers of the slice with dependency_id equal to the current value of dependency_id and quality_id equal to 0 that covers the macroblock with the macroblock address first_mb_in_slice when present in this slice: direct_spatial_mv_pred_flag, num_ref_idx_l0_active_minus1, num_ref_idx_l1_active_minus1, all syntax elements of the syntax structure ref_pic_list_reordering( ), base_pred_weight_table_flag, all syntax elements of the syntax structure pred_weight_table( ), all syntax elements of the syntax structure dec_ref_pic_marking( ), all syntax elements of the syntax structure dec_ref_base_pic_marking( ), slice_group_change_cycle, and store_ref_base_pic_flag.

**first_mb_in_slice** has the same semantics as specified in subclause 7.4.3.

**slice_type** specifies the coding type of the slice according to Table G-1.

**Table G-1 – Name association to slice_type for NAL units with nal_unit_type equal to 20.**

| slice_type | Name of slice_type |
|---|---|
| 0, 5 | EP (P slice in scalable extension) |
| 1, 6 | EB (B slice in scalable extension) |
| 2, 7 | EI (I slice in scalable extension) |

slice_type values in the range of 5 to 7, inclusive, specify, in addition to the coding type of the current slice, that all other slices of the current layer representation shall have a value of slice_type equal to the current value of slice_type or equal to the current value of slice_type minus 5.

When idr_flag is equal to 1 or num_ref_frames is equal to 0, slice_type shall be equal to 2 or 7.

**pic_parameter_set_id** has the same semantics as specified in subclause 7.4.3. The syntax element profile_idc of the SVC sequence parameter set that is referred to by the picture_parameter_set (by using seq_parameter_set_id) with pic_parameter_set_id shall be equal to 83 or 86.

**frame_num** is used as an identifier for dependency representations and shall be represented by log2_max_frame_num_minus4 + 4 bits in the bitstream.

frame_num is constrained as specified in subclause 7.4.3 where this constraint applies to dependency representations with dependency_id equal to the current value of dependency_id and the modifications a) and b) specified in subclause G.8.2 apply with currDependencyId being equal to the current value of dependency_id.

**field_pic_flag** and **bottom_field_flag** have the same semantics as specified in subclause 7.4.3.

**idr_pic_id** identifies an IDR picture when dependency_id is equal to the maximum present value of dependency_id in the VCL NAL units of the current access unit. The value of idr_pic_id shall be in the range of 0 to 65535, inclusive.

When dependency_id is equal to the maximum value of dependency_id in all VCL NAL units of the current access unit and two consecutive access units in decoding order are both IDR access units, the value of idr_pic_id in the slices of the dependency representation with dependency_id equal to the current value of dependency_id in the first such IDR access unit shall differ from the idr_pic_id in the slices of the dependency representation with dependency_id equal to the current value of dependency_id in the second such IDR access unit.

NOTE 1 – The classification of an access unit as IDR access unit depends on the maximum present value of dependency_id. When NAL units are removed from a bitstream, e.g. in order to adjust the bitstream to the capabilities of a receiving device, the classification of access units as IDR access units may change. Since all bitstreams for different conformance points supported in a scalable bitstream (especially for different maximum values of dependency_id) shall conform to this specification, the constraints on idr_pic_id shall be obeyed for all conformance points that are contained in a scalable bitstream.

**pic_order_cnt_lsb**, **delta_pic_order_cnt_bottom**, **delta_pic_order_cnt[** 0 ], and **delta_pic_order_cnt[** 1 ] have the same semantics as specified in subclause 7.4.3.

**redundant_pic_cnt** has the same semantics as specified in subclause 7.4.3.

**direct_spatial_mv_pred_flag** has the same semantics as specified in subclause 7.4.3. Additionally, the value of direct_spatial_mv_pred_flag shall be equal to 1.

**num_ref_idx_active_override_flag**, **num_ref_idx_l0_active_minus1**, and **num_ref_idx_l1_active_minus1** have the same semantics as specified in subclause 7.4.3.

**base_pred_weight_table_flag** equal to 1 specifies that the variables for weighted prediction are inferred. When base_pred_weight_table_flag is not present, it shall be inferred as follows.

– If no_inter_layer_pred_flag is equal to 1 or quality_id is greater than 0, base_pred_weight_table_flag is inferred to be equal to 0.

– Otherwise (no_inter_layer_pred_flag is equal to 0 and quality_id is equal to 0), base_pred_weight_table_flag is inferred to be equal to 1.

Let setOfRefLayerSlices be the set of slices that is represented by the VCL NAL units with dependency_id equal to (ref_layer_dq_id >> 4) and quality_id equal to 0 inside the current access unit.

When base_pred_weight_table_flag equal to 1 the following constraints shall be obeyed.

a) For all slices in setOfRefLayerSlices, the value of (slice_type % 5) shall be equal to (slice_type % 5) of the current slice.

b) When base_pred_weight_table_flag equal to 1 and the syntax structure pred_weight_table( ) is present in any slice in setOfRefLayerSlices, it shall be present for all slices in setOfRefLayerSlices, and all syntax elements inside the syntax structure pred_weight_table( ) shall be identical for all slices in setOfRefLayerSlices.

c) When base_pred_weight_table_flag is present in any slice in setOfRefLayerSlices, it shall have an identical value in all slices in setOfRefLayerSlices.

d) When the current slice is an EP slice, the following applies.

   i. The value of num_ref_idx_l0_active_minus1 of all slices in setOfRefLayerSlices shall be identical to the value of num_ref_idx_l0_active_minus1 of the current slice.

   ii. For each slice in setOfRefLayerSlices all syntax elements inside the syntax structure ref_pic_list_reordering( ) shall be identical, and the syntax structure ref_pic_list_reordering( ) for the slices in setOfRefLayerSlices shall contain syntax elements so that for useRefBasePicFlag equal to 0 and 1, an invocation of subclause G.8.2.3 with currDependencyId set equal to (ref_layer_dq_id >> 4),

useRefLayerRepFlag, and any slice of setOfRefLayerSlices as the inputs derives a reference picture list refPicList0RefLayer that is identical to the reference picture list refPicList0, which is derived by invoking subclause G.8.2.3 with currDependencyId set equal to dependency_id, useRefLayerRepFlag, and the current slice as the inputs.

iii. The syntax element weighted_pred_flag shall be equal to 1 for all slices in setOfRefLayerSlices.

e) When the current slice is an EB slice, the following applies.

i. The value of num_ref_idx_l0_active_minus1 and num_ref_idx_l1_active_minus1 of all slices in setOfRefLayerSlices shall be identical to the value of num_ref_idx_l0_active_minus1 and num_ref_idx_l1_active_minus1, respectively, of the current slice.

ii. For each slice in setOfRefLayerSlices all syntax elements inside the syntax structure ref_pic_list_reordering( ) shall be identical, and the syntax structure ref_pic_list_reordering( ) for the slices in setOfRefLayerSlices shall contain syntax elements so that for useRefBasePicFlag equal to 0 and 1, an invoking of subclause G.8.2.3 with currDependencyId set equal to (ref_layer_dq_id >> 4), useRefLayerRepFlag, and any slice of setOfRefLayerSlices as the inputs derives reference picture lists refPicList0RefLayer and refPicList1RefLayer that are identical to the reference picture lists refPicList0 and refPicList1, respectively, which are derived by invoking subclause G.8.2.3 with currDependencyId set equal to dependency_id, useRefLayerRepFlag, and the current slice as the inputs.

iii. The syntax element weighted_bipred_idc shall be equal to 1 for all slices in setOfRefLayerSlices.

When quality_id is greater than 0, all elements of pred_weight_table are inferred as specified in the beginning of this subclause.

**store_ref_base_pic_flag** has the same semantics as specified in subclause G.7.4.2.12.

**cabac_init_idc** and **slice_qp_delta** have the same semantics as specified in subclause 7.4.3.

**disable_deblocking_filter_idc** specifies whether the operation of the deblocking filter shall be disabled across some block edges of the slice and specifies for which edges the filtering is disabled. When disable_deblocking_filter_idc is not present in the slice header, the value of disable_deblocking_filter_idc shall be inferred to be equal to 0.

The value of disable_deblocking_filter_idc shall be in the range of 0 to 6, inclusive. disable_deblocking_filter_idc equal to 0 specifies that all luma and chroma block edges of the current picture are filtered. disable_deblocking_filter_idc equal to 1 specifies that deblocking is disabled for all block edges of the current picture. disable_deblocking_filter_idc equal to 2 specifies that all luma and chroma block edges are filtered with exception of the block edges that coincide with slice boundaries. disable_deblocking_filter_idc equal to 3 specifies a two stage deblocking filter process: After filtering all block luma and chroma block edges that do not coincide with slice boundaries (as if disable_deblocking_filter_idc were equal to 2), the luma and chroma block edges that coincide with slice boundaries are filtered. disable_deblocking_filter_idc equal to 4 specifies that all luma block edges of the current picture are filtered, but the deblocking of the chroma block edges is disabled. disable_deblocking_filter_idc equal to 5 specifies that all luma block edges are filtered with exception of the block edges that coincide with slice boundaries (as if disable_deblocking_filter_idc were equal to 2), and that deblocking for chroma block edges is disabled. disable_deblocking_filter_idc equal to 6 specifies that the deblocking for chroma block edges is disabled and that the two stage deblocking filter process is used for luma block edges: After filtering all block luma block edges that do not coincide with slice boundaries (as if disable_deblocking_filter_idc were equal to 2), the luma block edges that coincide with slice boundaries are filtered.

**slice_alpha_c0_offset_div2**, and **slice_beta_offset_div2** have the same semantics as specified in subclause 7.4.3.

**slice_group_change_cycle** has the same semantics as specified in subclause 7.4.3.

**ref_layer_dq_id** specifies the layer representation inside the current access unit that is used for inter-layer prediction of the current layer representation. When ref_layer_dq_id is not present, it shall be inferred as follows.

– If quality_id is greater than 0, ref_layer_dq_id is inferred to be equal to (DQId – 1).

– Otherwise (quality_id is equal to 0), ref_layer_dq_id is inferred to be equal to –1.

When quality_id is equal to 0, the NAL units with DQId equal to ref_layer_dq_id shall have discardable_flag equal to 0.

When present, the value of ref_layer_dq_id shall be the same for all coded slice NAL units of a layer representation.

When ref_layer_dq_id is not equal to –1, the following variables are derived as follows.

– RefLayerPicSizeInMbs is set equal to the value of the variable PicSizeInMbs for the layer representation with DQId equal to the value of ref_layer_dq_id.

– RefLayerPicWidthInMbs is set equal to the value of the variable PicWidthInMbs for the layer representation with DQId equal to the value of ref_layer_dq_id.

– RefLayerPicHeightInMbs is set equal to the value of the variable PicHeightInMbs for the layer representation with DQId equal to the value of ref_layer_dq_id.

– RefLayerChromaFormatIdc is set equal to the value of the syntax element chroma_format_idc for the layer representation with DQId equal to the value of ref_layer_dq_id.

– RefLayerChromaArrayType is set equal to the value of ChromaArrayType for the layer representation with DQId equal to the value of ref_layer_dq_id.

– $RefLayerPicWidthInSamples_L$ is set equal to the value of the variable $PicWidthInSamples_L$ for the layer representation with DQId equal to the value of ref_layer_dq_id.

– $RefLayerPicHeightInSamples_L$ is set equal to the value of the variable $PicHeightInSamples_L$ for the layer representation with DQId equal to the value of ref_layer_dq_id.

– $RefLayerPicWidthInSamples_C$ is set equal to the value of the variable $PicWidthInSamples_C$ for the layer representation with DQId equal to the value of ref_layer_dq_id.

– $RefLayerPicHeightInSamples_C$ is set equal to the value of the variable $PicHeightInSamples_C$ for the layer representation with DQId equal to the value of ref_layer_dq_id.

– RefLayerMbWidthC is set equal to the value of the variable MbWidthC for the layer representation with DQId equal to the value of ref_layer_dq_id.

– RefLayerMbHeightC is set equal to the value of the variable MbHeightC for the layer representation with DQId equal to the value of ref_layer_dq_id.

– RefLayerFrameMbsOnlyFlag is set equal to the value of the syntax element frame_mbs_only_flag for the layer representation with DQId equal to the value of ref_layer_dq_id.

– RefLayerFieldPicFlag is set equal to the value of the syntax element field_pic_flag for the layer representation with DQId equal to the value of ref_layer_dq_id.

– RefLayerBottomFieldFlag is set equal to the value of the syntax element bottom_field_flag for the layer representation with DQId equal to the value of ref_layer_dq_id.

– RefLayerMbaffFrameFlag is set equal to the value of the variable MbaffFrameFlag for the layer representation with DQId equal to the value of ref_layer_dq_id.

**disable_inter_layer_deblocking_filter_idc** specifies whether the operation of the deblocking filter for inter-layer intra prediction is disabled across some block edges of the slice and specifies for which edges the filtering is disabled. When disable_inter_layer_deblocking_filter_idc is not present in the slice header, the value of disable_inter_layer_deblocking_filter_idc shall be inferred to be equal to 0. The value of disable_inter_layer_deblocking_filter_idc shall be in the range of 0 to 6, inclusive. The values 0 to 6 of disable_inter_layer_deblocking_filter_idc specify the same deblocking filter operations as the corresponding values of disable_deblocking_filter_idc, but for the deblocking of the intra macroblocks of the dependency representation specified by ref_layer_dq_id before resampling.

When quality_id is equal to 0 and SpatialResolutionChangeFlag as specified in the following paragraphs is equal to 0, disable_inter_layer_deblocking_filter_idc shall be equal to 1.

**inter_layer_slice_alpha_c0_offset_div2** specifies the offset used in accessing the $\alpha$ and $t_{C0}$ deblocking filter tables for filtering operations controlled by the macroblocks within the slice for inter-layer prediction. From this value, the offset that is applied when addressing these tables shall be computed as:

$$InterlayerFilterOffsetA = inter\_layer\_slice\_alpha\_c0\_offset\_div2 << 1 \quad \text{(G-67)}$$

The value of inter_layer_slice_alpha_c0_offset_div2 shall be in the range of –6 to +6, inclusive. When inter_layer_slice_alpha_c0_offset_div2 is not present in the slice header, the value of inter_layer_slice_alpha_c0_offset_div2 shall be inferred to be equal to 0.

**inter_layer_slice_beta_offset_div2** specifies the offset used in accessing the $\beta$ deblocking filter table for filtering operations controlled by the macroblocks within the slice for inter-layer prediction. From this value, the offset that is applied when addressing the $\beta$ table of the deblocking filter is computed as:

$$InterlayerFilterOffsetB = inter\_layer\_slice\_beta\_offset\_div2 << 1 \quad \text{(G-68)}$$

The value of inter_layer_slice_beta_offset_div2 shall be in the range of –6 to +6, inclusive. When inter_layer_slice_beta_offset_div2 is not present in the slice header the value of inter_layer_slice_beta_offset_div2 shall be inferred to be equal to 0.

**constrained_intra_resampling_flag** specifies whether slice boundaries of the dependency representation that is specified for inter-layer prediction by ref_layer_dq_id are treated similar to layer picture boundaries for the intra resampling process. When constrained_intra_resampling_flag is equal to 1, disable_inter_layer_deblocking_filter_idc shall be equal to 1, 2, or 4.

When constrained_intra_resampling_flag is equal to 1, a macroblock cannot be coded using the Intra_Base macroblock prediction mode when it covers more than one slice in the reference layer as specified in subclause G.8.6.2.

When constrained_intra_resampling_flag is not present, it shall be inferred to be equal to 0.

**ref_layer_chroma_phase_x_plus1_flag** specifies the horizontal phase shift of the chroma components in units of half luma samples of a layer frame for the layer pictures that may be used for inter-layer prediction. When ref_layer_chroma_phase_x_plus1_flag is not present, it shall be inferred to be equal to seq_ref_layer_chroma_phase_x_plus1_flag.

When no_inter_layer_pred_flag is equal to 0, the following applies.

– When ref_layer_dq_id is greater than 0, ref_layer_chroma_phase_x_plus1_flag should be equal to chroma_phase_x_plus1_flag of the subset sequence parameter set RBSP that is referred to by the reference layer representation (with DQId equal to ref_layer_dq_id).

– When chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are present in the SVC sequence parameter set that is referred to by the reference layer representation (with DQId equal to ref_layer_dq_id), the following applies.

  – If ref_layer_chroma_phase_x_plus1_flag is equal to 0, chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field of the SVC sequence parameter set that is referred to by the reference layer representation should be equal to 0, 2, or 4.

  – Otherwise (ref_layer_chroma_phase_x_plus1_flag is equal to 1), chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field of the SVC sequence parameter set that is referred to by the reference layer representation should be equal to 1, 3, or 5.

**ref_layer_chroma_phase_y_plus1** specifies the vertical phase shift of the chroma components in units of half luma samples of a layer frame for the layer pictures that may be used for inter-layer prediction. When ref_layer_chroma_phase_y_plus1 is not present, it shall be inferred to be equal to seq_ref_layer_chroma_phase_y_plus1. The value of ref_layer_chroma_phase_y_plus1 shall be in the range of 0 to 2, inclusive.

When no_inter_layer_pred_flag is equal to 0, the following applies.

– When ref_layer_dq_id is greater than 0, ref_layer_chroma_phase_y_plus1 should be equal to chroma_phase_y_plus1 of the subset sequence parameter set RBSP that is referred to by the reference layer representation (with DQId equal to ref_layer_dq_id).

– When chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are present in the SVC sequence parameter set that is referred to by the reference layer representation (with DQId equal to ref_layer_dq_id), the following applies.

  – If ref_layer_chroma_phase_y_plus1 is equal to 0, chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field of the SVC sequence parameter set that is referred to by the reference layer representation should be equal to 2 or 3.

  – Otherwise, if ref_layer_chroma_phase_y_plus1 is equal to 1, chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field of the SVC sequence parameter set that is referred to by the reference layer representation should be equal to 0 or 1.

  – Otherwise (chroma_phase_y_plus1 is equal to 2), chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field of the SVC sequence parameter set that is referred to by the reference layer representation should be equal to 4 or 5.

**scaled_ref_layer_left_offset** specifies the horizontal offset between the upper-left luma sample of a resampled layer picture used for inter-layer prediction and the upper-left luma sample of the current picture or current layer picture in units of two luma samples.

When scaled_ref_layer_left_offset is not present, it shall be inferred as follows.

– If quality_id is greater than 0, scaled_ref_layer_left_offset is inferred to be equal to 0.

– Otherwise (quality_id is equal to 0), scaled_ref_layer_left_offset is inferred to be equal to seq_scaled_ref_layer_left_offset.

**scaled_ref_layer_top_offset** specifies the vertical offset between the upper-left luma sample of an resampled layer picture used for inter-layer prediction and the upper-left luma sample of the current picture or current layer picture. The vertical offset is specified in units of two luma samples when frame_mbs_only_flag is equal to 1, and it is specified in units of four luma samples when frame_mbs_only_flag is equal to 0.

When scaled_ref_layer_top_offset is not present, it shall be inferred as follows.

– If quality_id is greater than 0, scaled_ref_layer_top_offset is inferred to be equal to 0.

– Otherwise (quality_id is equal to 0), scaled_ref_layer_top_offset is inferred to be equal to seq_scaled_ref_layer_top_offset.

**scaled_ref_layer_right_offset** specifies the horizontal offset between the bottom-right luma sample of an resampled layer picture used for inter-layer prediction and the bottom-right luma sample of the current picture or current layer picture in units of two luma samples.

When scaled_ref_layer_right_offset is not present, it shall be inferred as follows.

– If quality_id is greater than 0, scaled_ref_layer_right_offset is inferred to be equal to 0.

– Otherwise (quality_id is equal to 0), scaled_ref_layer_right_offset is inferred to be equal to seq_scaled_ref_layer_right_offset.

**scaled_ref_layer_bottom_offset** specifies the vertical offset between the bottom-right luma sample of an resampled layer picture used for inter-layer prediction and the bottom-right luma sample of the current picture or current layer picture. The vertical offset is specified in units of two luma samples when frame_mbs_only_flag is equal to 1, and it is specified in units of four luma samples when frame_mbs_only_flag is equal to 0.

When scaled_ref_layer_bottom_offset is not present, it shall be inferred as follows.

– If quality_id is greater than 0, scaled_ref_layer_bottom_offset is inferred to be equal to 0.

– Otherwise (quality_id is equal to 0), scaled_ref_layer_bottom_offset is inferred to be equal to seq_scaled_ref_layer_bottom_offset.

The variables ScaledRefLayerLeftOffset, ScaledRefLayerRightOffset, ScaledRefLayerTopOffset, ScaledRefLayerBottomOffset, $ScaledRefLayerPicWidthInSamples_L$, and $ScaledRefLayerPicHeightInSamples_L$ are derived by

$$ScaledRefLayerLeftOffset = 2 * scaled\_ref\_layer\_left\_offset \quad (G\text{-}69)$$

$$ScaledRefLayerRightOffset = 2 * scaled\_ref\_layer\_right\_offset \quad (G\text{-}70)$$

$$ScaledRefLayerTopOffset = 2 * scaled\_ref\_layer\_top\_offset * ( 2 - frame\_mbs\_only\_flag ) \quad (G\text{-}71)$$

$$ScaledRefLayerBottomOffset = 2 * scaled\_ref\_layer\_bottom\_offset * ( 2 - frame\_mbs\_only\_flag ) \quad (G\text{-}72)$$

$$ScaledRefLayerPicWidthInSamples_L = PicWidthInMbs * 16 - ScaledRefLayerLeftOffset - ScaledRefLayerRightOffset \quad (G\text{-}73)$$

$$ScaledRefLayerPicHeightInSamples_L = PicHeightInMbs * 16 - ( ScaledRefLayerTopOffset + ScaledRefLayerBottomOffset ) / ( 1 + field\_pic\_flag ) \quad (G\text{-}74)$$

When no_inter_layer_pred_flag is equal to 0, the following applies.

– The bitstream shall not contain data that result in $ScaledRefLayerPicWidthInSamples_L$ less than $RefLayerPicWidthInSamples_L$.

– When RefLayerFrameMbsOnlyFlag is equal to 0 or frame_mbs_only_flag is equal to 1, the bitstream shall not contain data that result in ($ScaledRefLayerPicHeightInSamples_L$ * ( 1 + field_pic_flag )) less than ($RefLayerPicHeightInSamples_L$ * ( 1 + RefLayerFieldPicFlag )).

– When RefLayerFrameMbsOnlyFlag is equal to 1 and frame_mbs_only_flag is equal to 0, the bitstream shall not contain data that result in ($ScaledRefLayerPicHeightInSamples_L$ * ( 1 + field_pic_flag )) less than (2 * $RefLayerPicHeightInSamples_L$).

When ChromaArrayType is not equal to 0, the variables $ScaledRefLayerPicWidthInSamples_C$, and $ScaledRefLayerPicHeightInSamples_C$ are derived by

$$ScaledRefLayerPicWidthInSamples_C = ScaledRefLayerPicWidthInSamples_L / SubWidthC \quad (G\text{-}75)$$
$$ScaledRefLayerPicHeightInSamples_C = ScaledRefLayerPicHeightInSamples_L / SubHeightC \quad (G\text{-}76)$$

The variable CroppingChangeFlag is derived as follows.

– If quality_id is equal to 0 and extended_spatial_scalability is equal to 2, CroppingChangeFlag is set equal to 1.

– Otherwise (quality_id is greater than 0 or extended_spatial_scalability is less than 2), CroppingChangeFlag is set equal to 0.

The variable SpatialResolutionChangeFlag is derived as follows.

– If no_inter_layer_pred_flag is equal to 1, quality_id is greater than 0, or all of the following conditions are true, SpatialResolutionChangeFlag is set equal to 0.

  – CroppingChangeFlag is equal to 0

  – $ScaledRefLayerPicWidthInSamples_L$ is equal to $RefLayerPicWidthInSamples_L$

  – $ScaledRefLayerPicHeightInSamples_L$ is equal to $RefLayerPicHeightInSamples_L$

  – (ScaledRefLayerLeftOffset % 16) is equal to 0

  – (ScaledRefLayerTopOffset % ( 16 * ( 1 + frame_mbs_only_flag ) ) ) is equal to 0

  – field_pic_flag is equal to RefLayerFieldPicFlag

  – MbaffFrameFlag is equal to RefLayerMbaffFrameFlag

  – chroma_format_idc is equal to RefLayerChromaFormatIdc

  – chroma_phase_x_plus1_flag is equal to ref_layer_chroma_phase_x_plus1_flag

  – chroma_phase_y_plus1 is equal to ref_layer_chroma_phase_y_plus1

– Otherwise, SpatialResolutionChangeFlag is set equal to 1.

The variable RestrictedSpatialResolutionChangeFlag is derived as follows.

– If no_inter_layer_pred_flag is equal to 1, quality_id is greater than 0, or all of the following conditions are true, RestrictedSpatialResolutionChangeFlag is set equal to 1.

  – $ScaledRefLayerPicWidthInSamples_L$ is equal to $RefLayerPicWidthInSamples_L$ or (2 * $RefLayerPicWidthInSamples_L$)

  – $ScaledRefLayerPicHeightInSamples_L$ is equal to $RefLayerPicHeightInSamples_L$ or (2 * $RefLayerPicHeightInSamples_L$)

  – (ScaledRefLayerLeftOffset % 16) is equal to 0

  – (ScaledRefLayerTopOffset % ( 16 * ( 1 + frame_mbs_only_flag ) ) ) is equal to 0

  – MbaffFrameFlag is equal to RefLayerMbaffFrameFlag

  – field_pic_flag is equal to RefLayerFieldPicFlag

– Otherwise, RestrictedSpatialResolutionChangeFlag is set equal to 0.

**slice_skip_flag** equal to 0 specifies the presence of slice data in the slice. slice_skip_flag equal to 1 specifies that the syntax elements for the macroblock layer of the slice are derived by the following process.

1. CurrMbAddr is derived by

   $$CurrMbAddr = first\_mb\_in\_slice * ( 1 + MbaffFrameFlag ) \quad (G\text{-}77)$$

2. The variable mbIdx proceeds over the values 0..num_mbs_in_slice_minus1, and for each value of mbIdx, the following ordered steps are specified.

   a. The bitstream shall not contain data that result in in_crop_window( CurrMbAddr ) equal to 0.

b. For the macroblock with address CurrMbAddr, the syntax elements mb_skip_flag (when applicable), mb_skip_run (when applicable), mb_field_decoding_flag, base_mode_flag, residual_prediction_flag and coded_block_pattern shall be inferred as follows.

- mb_skip_flag (when applicable) and mb_skip_run (when applicable) are inferred to be equal to 0
- mb_field_decoding_flag is inferred to be equal to 0
  NOTE 2 – The frame/field mode used for decoding is inferred in subclause G.8.1.5.1.
- base_mode_flag is inferred to be equal to 1
- residual_prediction_flag is inferred to be equal to 1
- coded_block_pattern is inferred to be equal to 0
- $QP_Y$ is inferred to be equal to $SliceQP_Y$
- $QP'_Y$ is inferred to be equal to $(QP_Y + QpBdOffset_Y)$

c. CurrMbAddr is set to NextMbAddress( CurrMbAddr )

**num_mbs_in_slice_minus1** plus 1specifies the number of macroblocks for a slice with slice_skip_flag equal to 1.

**adaptive_base_mode_flag** specifies the presence of syntax elements in the slice header and in the macroblock layer in scalable extension. When adaptive_base_mode_flag is not present, it shall be inferred to be equal to 0.

**default_base_mode_flag** specifies how base_mode_flag is inferred when it is not present in macroblock layer in scalable extension. When default_base_mode_flag is not present, it shall be inferred to be equal to 0.

**adaptive_motion_prediction_flag** specifies the presence of syntax elements in the macroblock layer in scalable extension. When adaptive_motion_prediction_flag is not present, it shall be inferred to be equal to 0.

**default_motion_prediction_flag** specifies how motion_prediction_flag_l0[ ] and motion_prediction_flag_l1[ ] are inferred when they are not present in macroblock layer in scalable extension. When default_motion_prediction_flag is not present, it shall be inferred to be equal to 0.

**adaptive residual prediction flag** specifies the presence of syntax elements in the macroblock layer in scalable extension. When adaptive_residual_prediction_flag is not present, it shall be inferred to be equal to 0.

**default_residual_prediction_flag** specifies how residual_prediction_flag is inferred when it is not present in the macroblock layer in scalable extension. When default_residual_prediction_flag is not present, it shall be inferred to be equal to 0.

**tcoeff_level_prediction_flag** equal to 1 specifies that an alternative inter-layer prediction process is applied as specified in subclause G.8. When tcoeff_level_prediction_flag is not present it shall be inferred as follows.

- If no_inter_layer_pred_flag is equal to 1, tcoeff_level_prediction_flag is inferred to be equal to 0.
- Otherwise (no_inter_layer_pred_flag is equal to 0), tcoeff_level_prediction_flag is inferred to be equal to the value of seq_tcoeff_level_prediction_flag.

When SpatialResolutionChangeFlag is equal to 1, tcoeff_level_prediction_flag shall be equal to 0.

When tcoeff_level_prediction_flag is equal to 1, the following constraints shall be obeyed.

a) The layer representation with DQId equal to ref_layer_dq_id shall have no_inter_layer_pred_flag equal to 1 or tcoeff_level_prediction_flag equal to 1.

b) All elements of ScalingList4x4 shall be the same for the slices of the current layer representation and all slices of the layer representation with DQId equal to the value of ref_layer_dq_id.

c) All elements of ScalingList8x8 shall be the same for the slices of the current layer representation and all slices of the layer representation with DQId equal to the value of ref_layer_dq_id.

d) The value of the syntax element use_ref_base_pic_flag shall be equal to 0 for the slices of the current layer representation and all slices of the layer representation with DQId equal to the value of ref_layer_dq_id.

**scan_idx_start** specifies the first scanning position for the transform coefficient levels in the current slice. When scan_idx_start is not present, it shall be inferred to be equal to 0.

**scan_idx_end** specifies the last scanning position for the transform coefficient levels in the current slice. When scan_idx_end is not present, it shall be inferred to be equal to 15.

#### G.7.4.3.5 Decoded reference base picture marking semantics

The syntax elements adaptive_ref_base_pic_marking_mode_flag, memory_management_control_operation, difference_of_pic_nums_minus1 and long_term_pic_num specify marking of reference base pictures as "unused for reference".

**adaptive_ref_base_pic_marking_mode_flag** has the same semantics as the syntax element adaptive_ref_pic_marking_mode_flag in the dec_ref_pic_marking() syntax structure. When adaptive_ref_base_pic_marking_mode_flag is not present, it shall be inferred to be equal to 0.

**memory_management_control_operation** specifies a control operation to be applied to affect the reference picture marking. The memory_management_control_operation syntax element is followed by data necessary for the operation specified by the value of memory_management_control_operation. The values and control operations associated with memory_management_control_operation are specified in Table G-2. The memory_management_control_operation syntax elements are processed by the decoding process in the order in which they appear, and the semantics constraints expressed for each memory_management_control_operation apply at the specific position in that order at which that individual memory_management_control_operation is processed.

memory_management_control_operation shall not be equal to 1 unless the specified reference picture is marked as "used for short-term reference" and as "reference base picture" when the memory_management_control_operation is processed by the decoding process.

memory_management_control_operation shall not be equal to 2 unless the specified long-term picture number refers to a reference picture that is marked as "used for long-term reference" and as "reference base picture" when the memory_management_control_operation is processed by the decoding process.

**Table G-2 – Memory management control operation (memory_management_control_operation) values**

| memory_management_control_operation | Memory Management Control Operation |
|---|---|
| 0 | End memory_management_control_operation syntax element loop |
| 1 | Mark a short-term reference base picture as "unused for reference" |
| 2 | Mark a long-term reference base picture as "unused for reference" |

**difference_of_pic_nums_minus1** is used (with memory_management_control_operation equal to 1) to mark a short-term reference picture as "unused for reference". When the associated memory_management_control_operation is processed by the decoding process, the resulting picture number derived from difference_of_pic_nums_minus1 shall be a picture number assigned to one of the reference pictures marked as "used for reference" and as "reference base picture".

The resulting picture number is constrained as follows.

– If field_pic_flag is equal to 0, the resulting picture number shall be one of the set of picture numbers assigned to reference frames or complementary reference field pairs, or there is no reference frame or complementary reference field pair that has been assigned the resulting picture number.

– Otherwise (field_pic_flag is equal to 1), the resulting picture number shall be one of the set of picture numbers assigned to reference fields, or there is no reference pair that has been assigned the resulting picture number.

**long_term_pic_num** is used (with memory_management_control_operation equal to 2) to mark a long-term reference picture as "unused for reference". When the associated memory_management_control_operation is processed by the decoding process, long_term_pic_num shall be equal to a long-term picture number assigned to one of the reference pictures marked as "used for long-term reference" and as "reference base picture".

The resulting long-term picture number is constrained as follows.

– If field_pic_flag is equal to 0, the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference frames or complementary reference field pairs, or there is no reference frame or complementary reference field pair that has been assigned the resulting long-term picture number..

– Otherwise (field_pic_flag is equal to 1), the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference fields, or there is no reference pair that has been assigned the resulting long-term picture number.

When present in a prefix NAL unit, the syntax elements of the dec_ref_base_pic_marking( ) syntax structure are considered to apply to the associated NAL unit.

#### G.7.4.4 Slice data semantics

The semantics specified in subclause 7.4.4 apply.

##### G.7.4.4.1 Slice data in scalable extension semantics

The semantic specified in subclauses 7.4.4 apply with the following modifications.

**mb_skip_run** specifies the number of consecutive skipped macroblocks for which, when decoding an EP slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as a P macroblock type, or for which, when decoding an EB slice, mb_type shall be inferred to be B_Skip and the macroblock type is collectively referred to as a B macroblock type. The value of mb_skip_run shall be in the range of 0 to PicSizeInMbs – CurrMbAddr, inclusive.

**mb_skip_flag** equal to 1 specifies that for the current macroblock, when decoding an EP slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as P macroblock type, or for which, when decoding an EB slice, mb_type shall be inferred to be B_Skip and the macroblock type is collectively referred to as B macroblock type. mb_skip_flag equal to 0 specifies that the current macroblock is not skipped.

#### G.7.4.5 Macroblock layer semantics

The semantics specified in subclause 7.4.5 apply. Additionally, the following applies.

The macroblock_layer( ) syntax structure shall be considered to contain the following syntax elements with the following inferred values.

– base_mode_flag is inferred to be equal to 0.

– residual_prediction_flag is inferred to be equal to 0.

##### G.7.4.5.1 Macroblock prediction semantics

The semantics specified in subclause 7.4.5.1 apply. Additionally, the following applies.

The mb_pred( ) syntax structure shall be considered to contain the following syntax elements with the following inferred values.

– motion_prediction_flag_l0[ mbPartIdx ] is inferred to be equal to 0 for each value of mbPartIdx in the range from 0 to NumMbPart( mb_type ) − 1, inclusive.

– motion_prediction_flag_l1[ mbPartIdx ] is inferred to be equal to 0 for each value of mbPartIdx in the range from 0 to NumMbPart( mb_type ) − 1, inclusive.

##### G.7.4.5.2 Sub-macroblock prediction semantics

The semantics specified in subclause 7.4.5.2 apply. Additionally, the following applies.

The sub_mb_pred( ) syntax structure shall be considered to contain the following syntax elements with the following inferred values.

– motion_prediction_flag_l0[ mbPartIdx ] is inferred to be equal to 0 for each value of mbPartIdx in the range from 0 to 3, inclusive.

– motion_prediction_flag_l1[ mbPartIdx ] is inferred to be equal to 0 for each value of mbPartIdx in the range from 0 to 3, inclusive.

##### G.7.4.5.3 Residual data semantics

The semantics specified in subclause 7.4.5.3 apply.

###### G.7.4.5.3.1 Residual luma semantics

The semantics specified in subclause 7.4.5.3.1 apply.

###### G.7.4.5.3.2 Residual block CAVLC semantics

The semantics specified in subclause 7.4.5.3.2 apply.

###### G.7.4.5.3.3 Residual block CABAC semantics

The semantics specified in subclause 7.4.5.3.3 apply.

### G.7.4.6 Macroblock layer in scalable extension semantics

The semantic specified in subclauses 7.4.5 apply. Additionally, the following modifications and extensions are specified.

**base_mode_flag** equal to 1 specifies that the macroblock prediction mode as well as the corresponding motion data (when applicable) are inferred as specified in subclause G.8. base_mode_flag equal to 0 specifies that mb_type is not inferred.

When base_mode_flag is not present, base_mode_flag shall be inferred as follows.

– If in_crop_window( CurrMbAddr ) is equal to 0, the value of base_mode_flag is inferred to be equal to 0.

– Otherwise (in_crop_window( CurrMbAddr ) is equal to 1), the value of base_mode_flag is inferred to be equal to default_base_mode_flag.

When store_ref_base_pic_flag is equal to 1 and quality_id is greater than 0, base_mode_flag shall be equal to 1.

**mb_type** specifies the macroblock type. The semantics of mb_type depend on the slice type.

Tables and semantics are specified for the various macroblock types for EI, EP, and EB slices. Each table presents the value of mb_type, the name of mb_type, the number of macroblock partitions used (given by NumMbPart( mb_type ) function), the prediction mode of the macroblock (when it is not partitioned) or the first partition (given by the MbPartPredMode( mb_type, 0 ) function) and the prediction mode of the second partition (given by the MbPartPredMode( mb_type, 1 ) function). When a value is not applicable it is designated by "na". In the text, the value of mb_type may be referred to as the macroblock type and a value X of MbPartPredMode( ) may be referred to in the text by "X macroblock (partition) prediction mode" or as "X prediction macroblocks".

Table G-3 shows the allowed collective macroblock types for each slice_type.

**Table G-3 – Allowed collective macroblock types for slice_type.**

| slice_type | allowed collective macroblock types |
|---|---|
| EI (slice) | I (see Table 7-11 and Table G-4) (macroblock types) |
| EP (slice) | P (see Table 7-13) and I (see Table 7-11 and Table G-4) (macroblock types) |
| EB (slice) | B (see Table 7-14) and I (see Table 7-11 and Table G-4) (macroblock types) |

Macroblock types that may be collectively referred to as I macroblock types are specified in Table G-4 and Table 7-11. mb_type value 0 is specified in Table G-4, whereas the other mb_type are specified in Table 7-11. Table G-4 additionally specifies the inferred macroblock type I_BL.

The macroblock types for EI slices are all I macroblock types.

**Table G-4 – Macroblock types for EI slices.**

| mb_type | Name of mb_type | transform_size_8x8_flag | MbPartPredMode ( mb_type, 0 ) | Intra16x16PredMode | CodedBlockPatternChroma | CodedBlockPatternLuma |
|---|---|---|---|---|---|---|
| 0 | I_NxN | 0 | Intra_4x4 | na | Equation 7-34 | Equation 7-34 |
| 0 | I_NxN | 1 | Intra_8x8 | na | Equation 7-34 | Equation 7-34 |
| inferred | I_BL | na | Intra_Base | na | Equation 7-34 | Equation 7-34 |

Macroblock types that may be collectively referred to as P macroblock types are specified in Table 7-13.

The macroblock types for EP slices are specified in Table 7-13, Table 7-11 and Table G-4. mb_type values 0 to 4 are specified in Table 7-13, mb_type value 5 is specified in Table G-4 indexed by subtracting 5 from the value of mb_type and mb_type values 6 to 30 are specified in Table 7-11, indexed by subtracting 5 from the value of mb_type. Table G-4 additionally specifies the inferred macroblock type I_BL.

Macroblock types that may be collectively referred to as B macroblock types are specified in Table 7-14.

The macroblock types for EB slices are specified in Table 7-14, Table 7-11 and Table G-4. mb_type values 0 to 22 are specified in Table 7-14, mb_type values 23 is specified in Table G-4 indexed by subtracting 23 from the value of mb_type and mb_type 24 to 48 are specified in Table 7-14, indexed by subtracting 23 from the value of mb_type. Table G-4 additionally specifies the inferred macroblock type I_BL.

**coded_block_pattern** specifies which of the four 8x8 luma blocks and associated chroma blocks of a macroblock may contain non-zero transform coefficient values. For macroblocks with prediction mode not equal to Intra_16x16, coded_block_pattern is present in the bitstream and the variables CodedBlockPatternLuma and CodedBlockPatternChroma are derived as specified by Equation 7-34.

The meaning of CodedBlockPatternLuma and CodedBlockPatternChroma is specified in subclause 7.4.5.

**residual_prediction_flag** equal to 1 specifies that the residual signal of the current macroblock is predicted as specified in subclause G.8 using the layer representation specified by ref_layer_dq_id. residual_prediction_flag equal to 0 specifies that the residual signal of the current macroblock is not predicted.

When the syntax element residual_prediction_flag is not present, residual_prediction_flag shall be inferred as follows.

– If all of the following conditions are true, residual_prediction_flag is inferred to be equal to default_residual_prediction_flag.
  – slice_type is not equal to EI
  – in_crop_window( CurrMbAddr ) is equal to 1
  – base_mode_flag is equal to 1 or mb_type does not specify an I macroblock type
– Otherwise, residual_prediction_flag is inferred to be equal to 0.

All elements of the arrays LumaLevel, LumaLevel8x8, Intra16x16DCLevel, Intra16x16ACLevel, CbLevel, CbLevel8x8, CbIntra16x16DCLevel, CbIntra16x16ACLevel, CrLevel, CrLevel8x8, CrIntra16x16DCLevel, CrIntra16x16ACLevel, ChromaDCLevel, and ChromaACLevel are set equal to 0 before parsing the residual( ) syntax structure.

#### G.7.4.6.1 Macroblock prediction in scalable extension semantics

The semantics specified in subclauses 7.3.5.1 apply. Additionally, the following semantics are specified.

**motion_prediction_flag_l0[** mbPartIdx **]** equal to 1 specifies that an alternative motion vectors prediction process as specified in subclause G.8 is used for deriving the list 0 motion vector of the macroblock partition mbPartIdx and that the list 0 reference index of the macroblock partition mbPartIdx is inferred as specified in subclause G.8.

When motion_prediction_flag_l0[ mbPartIdx ] is not present, motion_prediction_flag_l0[ mbPartIdx ] shall be inferred as follows.

– If in_crop_window( CurrMbAddr ) is equal to 0, motion_prediction_flag_l0[ mbPartIdx ] is inferred to be equal to 0.
– Otherwise (in_crop_window( CurrMbAddr ) is equal to 1), motion_prediction_flag_l0[ mbPartIdx ] is inferred to be equal to default motion prediction flag.

**motion_prediction_flag_l1[** mbPartIdx **]** has the same semantics as motion_prediction_flag_l0[ mbPartIdx ], with l0 and list 0 replaced by l1 and list 1, respectively.

#### G.7.4.6.2 Sub-macroblock prediction in scalable extension semantics

The semantic specified in subclauses 7.4.5.2 apply. Additionally, the following semantics are specified.

**motion_prediction_flag_l0[** mbPartIdx **]** and **motion_prediction_flag_l1[** mbPartIdx **]** have the same semantics as specified in subclause G.7.3.6.1.

## G.8 SVC decoding process

This subclause describes the decoding process for an access unit, given syntax elements and upper-case variables from subclause G.7 (with reference made to clause 7 in subclause G.7) that are derived from the bitstream.

NOTE 1 – All syntax elements and upper-case variables from subclause G.7 are available for the entire current access unit. When syntax elements or upper-case variables appear with identical names in subclause G.7 they are referred herein through unique identifiers.

Output of this process are decoded samples of the current primary coded picture.

The decoding process is specified such that all decoders shall produce numerically identical results. Any decoding process that produces identical results to the process described here conforms to the decoding process requirements of this Recommendation | International Standard.

All sub-bitstreams that can be derived using the sub-bitstream extraction process as specified in subclause G.8.8.1 with any combination of values for priority_id, temporal_id, dependency_id, or quality_id as the input shall result in a set of coded video sequences, with each coded video sequence conforming to one or more of the profiles specified in Annex A and Annex G.

This subclause specifies the decoding process for an access unit of a coded video sequence conforming to one or more of the profiles specified in subclause G.10.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the decoding process specified in this subclause and all child processes invoked from the process specified in this subclause are the syntax elements and derived upper-case variables for the current access unit.

At the start of the decoding process for an access unit, the following applies.

1. Variables and functions relating to picture order count are derived by invoking the SVC decoding process for picture order count as specified in subclause G.8.2.1.
2. The SVC decoding process for gaps in frame_num as specified in subclause G.8.2.5 is invoked.
3. For each value of dependency_id present in the NAL units that are input to this process as specified above, the following applies.
   - The decoding process for macroblock to slice group map as specified in subclause 8.2.2 is invoked with the syntax elements of the NAL units with dependency_id as input.
   - The function NextMbAddress( ) as specified in subclause 8.2.2 is used for parsing the slice data syntax structures of all NAL units with the corresponding value of dependency_id.

The derivation process for the set of layer representations required for decoding as specified in subclause G.8.1.1 is invoked and the output is a list dqIdList of integer values specifying layer representation identifiers. The variables DQIdMin and DQIdMax are set equal to the minimum and maximum values, respectively, of the entries of the list dqIdList, and the variable DependencyIdMax is set equal to (DQIdMax >> 4). DependencyIdMax shall be the same for all access units of the coded video sequence.

The collective terms currentVars and refLayerVars are initially marked as not available.

The variable currDQId proceeds over the values DQIdMin..DQIdMax, and when a value of currDQId is present in the list dqIdList, the following ordered steps apply.

1. The variable spatResChangeFlag is set equal to the variable SpatialResolutionChangeFlag of the layer representation with DQId equal to currDQId.
2. Depending on spatResChangeFlag, the following applies.
   - If spatResChangeFlag is equal to 0, the base decoding process for layer representations without resolution change as specified in subclause G.8.1.3.1 is invoked with currDQId and currentVars as the inputs and the output is a modified version of currentVars.
   - Otherwise (spatResChangeFlag is equal to 1), the base decoding process for layer representations with resolution change as specified in subclause G.8.1.3.2 is invoked with currDQId and currentVars as the inputs and the outputs are variables assigned to the collective term refLayerVars and a modified version of currentVars.
3. When currDQId is equal to (DependencyIdMax << 4) and store_ref_base_pic_flag for the dependency representation with dependency_id equal to DependencyIdMax is equal to 1, the target layer representation decoding process as specified in subclause G.8.1.3.3 is invoked with currDQId, refLayerVars, and currentVars

as the inputs and the outputs are assigned to the sample array $B_L$ and, when ChromaArrayType is not equal to 0, the sample arrays $B_{Cb}$ and $B_{Cr}$.

NOTE 2 – The sample arrays $B_L$, $B_{Cb}$, and $B_{Cr}$ represent the reference base picture for an access unit with store_ref_base_pic_flag equal to 1 for the dependency representation with dependency_id equal to DependencyIdMax.

The target layer representation decoding process as specified in subclause G.8.1.3.3 is invoked with currDQId, refLayerVars, and currentVars as the inputs and the outputs are assigned to the sample array $S_L$ and, when ChromaArrayType is not equal to 0, the sample arrays $S_{Cb}$ and $S_{Cr}$.

NOTE 3 – The sample arrays $S_L$, $S_{Cb}$, and $S_{Cr}$ represent the decoded picture for the access unit.

The SVC decoded reference picture marking process as specified in subclause G.8.2.4 is invoked.

### G.8.1 SVC initialisation and decoding processes

Subclause G.8.1.1 specifies the derivation process for the set of layer representations required for decoding.

Subclause G.8.1.2 specifies the array assignment, initialisation, and restructuring processes.

Subclause G.8.1.3 specifies the layer representation decoding processes.

Subclause G.8.1.4 specifies the slice decoding processes.

Subclause G.8.1.5 specifies the macroblock initialisation and decoding processes.

#### G.8.1.1 Derivation process for the set of layer representations required for decoding

Inputs to this process are the coded slice NAL units of an access unit.

Output of this process is a list dqIdList of integer values specifying layer representation identifiers.

With currDQId being set equal to the maximum value of DQId for all coded slice NAL units of the access unit and with refLayerDQId( dqId ) specifying the maximum value of ref_layer_dq_id for all coded slice NAL units of the layer representation of the access unit with DQId equal to dqId, the list dqIdList is derived as specified by the following pseudo-code.

```
numEntries = 0
dqIdList[ numEntries++ ] = currDQId
while( refLayerDQId( currDQId ) >= 0 )  {
    dqIdList[ numEntries++ ] = refLayerDQId( currDQId )                    (G-78)
    currDQId = dqIdList[ numEntries – 1 ]
}
```

#### G.8.1.2 Array assignment, initialisation, and restructuring processes

Subclause G.8.1.2.1 specifies the array assignment and initialisation process.

Subclause G.8.1.2.2 specifies the array restructuring process.

##### G.8.1.2.1 Array assignment and initialisation process

Output of this process is a set of arrays that are assigned to the collective term currentVars.

The following arrays are collectively referred to as currentVars.

– A one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as sliceIdc[ mbAddr ]. All elements of the array sliceIdc are initially marked as unspecified.

– A one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of a layer representation are field macroblocks and which macroblocks are frame macroblocks. An element of this array for a macroblock with address mbAddr is referred to as fieldMbFlag[ mbAddr ]. All elements of the array fieldMbFlag are initially marked as unspecified.

– A one-dimensional array cTrafo with PicSizeInMbs elements specifying the luma and, when ChromArrayType is equal to 3, chroma transform types for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as cTrafo[ mbAddr ]. Unless marked as unspecified, each element of cTrafo is equal to T_4x4, T_8x8, T_16x16, or T_PCM. All elements of the array cTrafo are initially marked as unspecified.

– A one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as mbType[ mbAddr ]. Unless marked as unspecified, each element of mbType is equal to I_4x4, I_8x8, I_16x16, I_PCM, I_BL, or one of the Inter macroblock types specified in Table 7-13 and Table 7-14. All elements of the array mbType are initially marked as unspecified.

– A (PicSizeInMbs)x4 array subMbType specifying sub-macroblock types for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr and a macroblock partition index mbPartIdx is referred to as subMbType[ mbAddr ][ mbPartIdx ]. A one-dimensional array specifying sub-macroblock types for the macroblock partitions of a macroblock with address mbAddr is referred to as subMbType[ mbAddr ]. Unless marked as unspecified, each element of subMbType is equal to one of the sub-macroblock types specified in Table 7-17 and Table 7-18. All elements of the array subMbType are initially marked as unspecified.

– A one-dimensional array mvCnt with PicSizeInMbs elements specifying the number of motion vectors for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as mvCnt[ mbAddr ]. All elements of the array mvCnt are initially set equal to 0.

– A one-dimensional array $tQP_Y$ with PicSizeInMbs elements specifying luma quantisation parameters for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as $tQP_Y$[ mbAddr ]. All elements of the array $tQP_Y$ are initially set equal to 0.

– When ChromaArrayType is not equal to 0, two one-dimensional arrays $tQP_{Cb}$ and $tQP_{Cr}$ with PicSizeInMbs elements specifying Cb and Cr quantisation parameters, respectively, for the macroblocks of a layer representation. An element of these arrays for a macroblock with address mbAddr is referred to as $tQP_{CX}$[ mbAddr ] with CX being replaced by Cb or Cr. All elements of the arrays $tQP_{Cb}$ and $tQP_{Cr}$ are initially set equal to 0.

– A (PicSizeInMbs)x16 array ipred4x4 specifying Intra_4x4 prediction modes for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr and a 4x4 block with index c4x4BlkIdx is referred to as ipred4x4[ mbAddr ][ c4x4BlkIdx ]. A one-dimensional array specifying Intra_4x4 prediction modes for the 4x4 blocks of a macroblock with address mbAddr is referred to as ipred4x4[ mbAddr ]. All elements of the array ipred4x4 are initially marked as unspecified.

– A (PicSizeInMbs)x4 array ipred8x8 specifying Intra_8x8 prediction modes for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr and a 8x8 block with index c8x8BlkIdx is referred to as ipred8x8[ mbAddr ][ c8x8BlkIdx ]. A one-dimensional array specifying Intra_8x8 prediction modes for the 8x8 blocks of a macroblock with address mbAddr is referred to as ipred8x8[ mbAddr ]. All elements of the array ipred8x8 are initially marked as unspecified.

– A one-dimensional array ipred16x16 with PicSizeInMbs elements specifying Intra_16x16 prediction modes for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as ipred16x16[ mbAddr ]. All elements of the array ipred16x16 are initially marked as unspecified.

– When ChromaArrayType is equal to 1 or 2, a one-dimensional array ipredChroma with PicSizeInMbs elements specifying intra chroma prediction modes for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr is referred to as ipredChroma[ mbAddr ]. All elements of the array ipredChroma are initially marked as unspecified.

– Two (PicSizeInMbs)x4 arrays predFlagL0 and predFlagL1 specifying prediction utilization flags for the macroblocks of a layer representation. An element of these arrays for a macroblock with address mbAddr and a macroblock partition index mbPartIdx is referred to as predFlagLX[ mbAddr ][ mbPartIdx ] with X being replaced by 0 or 1. A one-dimensional array specifying prediction utilization flags for the macroblock partitions of a macroblock with address mbAddr is referred to as predFlagLX[ mbAddr ] with X being replaced by 0 or 1. All elements of the arrays predFlagL0 and predFlagL1 are initially set equal to 0.

– Two (PicSizeInMbs)x4 arrays refIdxL0 and refIdxL1 specifying reference indices for the macroblocks of a layer representation. An element of these arrays for a macroblock with address mbAddr and a macroblock partition index mbPartIdx is referred to as refIdxLX[ mbAddr ][ mbPartIdx ] with X being replaced by 0 or 1. A one-dimensional array specifying reference indices for the macroblock partitions of a macroblock with address mbAddr is referred to as refIdxLX[ mbAddr ] with X being replaced by 0 or 1. All elements of the arrays refIdxL0 and refIdxL1 are initially set equal to –1.

– Two (PicSizeInMbs)x4x4x2 arrays mvL0 and mvL1 specifying motion vector components for the macroblocks of a layer representation. An element of these arrays for a macroblock with address mbAddr, a macroblock partition index mbPartIdx, a sub-macroblock partition index subMbPartIdx, and a motion vector component index c is referred to as mvLX[ mbAddr ][ mbPartIdx ][ subMbPartIdx ][ c ] with X being replaced by 0 or 1. A one-dimensional array with 2 elements representing the motion vector for a sub-macroblock partition

subMbPartIdx of a macroblock partition mbPartIdx inside a macroblock mbAddr is referred to as mvLX[ mbAddr ][ mbPartIdx ][ subMbPartIdx ] with X being replaced by 0 or 1. A 4x2 array representing the motion vectors for a macroblock partition mbPartIdx inside a macroblock mbAddr is referred to as mvLX[ mbAddr ][ mbPartIdx ] with X being replaced by 0 or 1. A 4x4x2 array representing the motion vectors for a macroblock mbAddr is referred to as mvLX[ mbAddr ] with X being replaced by 0 or 1. A motion vector component with component index c for a macroblock partition mbPartIdx of a macroblock mbAddr that is not split into sub-macroblock partitions can also be referred to as mvLX[ mbAddr ][ mbPartIdx ][ c ] with X being replaced by 0 or 1, which is identical to mvLX[ mbAddr ][ mbPartIdx ][ 0 ][ c ]. A motion vector for a macroblock partition mbPartIdx of a macroblock mbAddr that is not split into sub-macroblock partitions can also be referred to as mvLX[ mbAddr ][ mbPartIdx ] with X being replaced by 0 or 1, which is identical to mvLX[ mbAddr ][ mbPartIdx ][ 0 ]. All elements of the arrays mvL0 and mvL1 are initially set equal to 0.

– A (PicSizeInMbs)x(256 + 2 * MbWidthC * MbHeightC) array tCoeffLevel specifying transform coefficient level values for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr and a transform coefficient index tCoeffIdx is referred to as tCoeffLevel[ mbAddr ][ tCoeffIdx ]. A one-dimensional array specifying the transform coefficient level values for a macroblock with address mbAddr is referred to as tCoeffLevel[ mbAddr ]. All elements of the array tCoeffLevel are initially set equal to 0.

– A (PicSizeInMbs)x(256 + 2 * MbWidthC * MbHeightC) array sTCoeff specifying scaled transform coefficient values for the macroblocks of a layer representation. An element of this array for a macroblock with address mbAddr and a transform coefficient index tCoeffIdx is referred to as sTCoeff[ mbAddr ][ tCoeffIdx ]. A one-dimensional array specifying the scaled transform coefficient values for a macroblock with address mbAddr is referred to as sTCoeff[ mbAddr ]. All elements of the array sTCoeff are initially set equal to 0.

– A ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array $rS_L$ specifying residual luma sample values for a layer picture. An element of this array for a luma location ( x, y ) relative to the upper-left luma sample of the macroblock with address 0 is referred to as $rS_L$[ x, y ]. All elements of the array $rS_L$ are initially set equal to 0.

– When ChromaArrayType is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $rS_{Cb}$ and $rS_{Cr}$ specifying residual chroma sample values for a layer picture. An element of these arrays for a chroma location ( x, y ) relative to the upper-left chroma sample of the macroblock with address 0 is referred to as $rS_{CX}$[ x, y ] with CX being replaced by Cb or Cr. All elements of the arrays $rS_{Cb}$ and $rS_{Cr}$ are initially set equal to 0.

– A ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array $cS_L$ specifying constructed luma sample values for a layer picture. An element of this array for a luma location ( x, y ) relative to the upper-left luma sample of the macroblock with address 0 is referred to as $cS_L$[ x, y ]. All elements of the array $cS_L$ are initially set equal to 0.

– When ChromaArrayType is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $cS_{Cb}$ and $cS_{Cr}$ specifying constructed chroma sample values for a layer picture. An element of these arrays for a chroma location ( x, y ) relative to the upper-left chroma sample of the macroblock with address 0 is referred to as $cS_{CX}$[ x, y ] with CX being replaced by Cb or Cr. All elements of the arrays $cS_{Cb}$ and $cS_{Cr}$ are initially set equal to 0.

#### G.8.1.2.2 Array restructuring process

This process is only invoked when no_inter_layer_pred_flag is equal to 0, SpatialResolutionChangeFlag is equal to 0, and any of the variables ScaledRefLayerLeftOffset, ScaledRefLayerRightOffset, ScaledRefLayerTopOffset, or ScaledRefLayerBottomOffset is not equal to 0.

Input to this process is a set of arrays collectively referred to as currentVars.

Output of this process is the set of arrays collectively referred to as currentVars with modifications related to the array sizes as well as the ordering of array elements.

The variables that are assigned to the collective term currentVars are assigned to the collective term refLayerVars.

The array assignment and initialisation process as specified in subclause G.8.1.2.1 is invoked and the output is the set of arrays collectively referred to as currentVars.

The variables xOffset, yOffset, xOffsetC, and yOffsetC are derived by

xOffset = ScaledRefLayerLeftOffset (G-79)
yOffset = ScaledRefLayerTopOffset / ( 1 + field_pic_flag ) (G-80)
xOffsetC = ( xOffset >> 4 ) * MbWidthC (G-81)
yOffsetC = ( yOffset >> 4 ) * MbHeightC (G-82)

For the macroblock address mbAddr proceeding over the values 0..(PicSizeInMbs – 1), the following ordered steps are specified.

1. With eS set equal to (1 + MbaffFrameFlag), the variables refMbX and refMbY are derived by

$$refMbX = ( ( mbAddr / eS ) \% PicWidthInMbs ) - ( xOffset / 16 ) \quad (G\text{-}83)$$
$$refMbY = ( ( mbAddr / eS ) / PicWidthInMbs ) * eS + ( mbAddr \% eS ) - ( yOffset / 16 ) \quad (G\text{-}84)$$

2. The reference layer macroblock address refMbAddr is derived as follows.
   - If any of the following conditions is true, refMbAddr is marked as not available.
     - refMbX is less than 0 or refMbX is greater than or equal to RefLayerPicWidthInMbs
     - refMbY is less than 0 or refMbY is greater than or equal to RefLayerPicHeightInMbs
   - Otherwise, with bS set equal to (1 + RefLayerMbaffFrameFlag), refMbAddr is derived by

$$refMbAddr = ( refMbY / bS ) * bS * RefLayerPicWidthInMbs + ( refMbY \% bS ) + refMbX \quad (G\text{-}85)$$

3. When refMbAddr is available, for X being replaced by sliceIdc, fieldMbFlag, cTrafo, mbType, subMbType, mvCnt, $tQP_Y$, $tQP_{Cb}$ (when ChromaArrayType is not equal to 0), $tQP_{Cb}$ (when ChromaArrayType is not equal to 0), ipred4x4, ipred8x8, ipred16x16, ipredChroma (when ChromaArrayType is equal to 1 or 2), predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, tCoeffLevel, and sTCoeff and with currArray representing the array X of the collective term currentVars and refLayerArray representing the array X of the collective term refLayerVars, the array element currArray[ mbAddr ], which can be a scalar or an array, is set equal to the array element refLayerArray[ refMbAddr ].

For X being replaced by $rS_Y$ and $cS_Y$ and with currArray representing the array X of the collective term currentVars and refLayerArray representing the array X of the collective term refLayerVars, the array currArray is modified by

$$currArray[ x, y ] = refLayerArray[ x - xOffset, y - yOffset ] \quad (G\text{-}86)$$
$$\text{with } x = Max( 0, xOffset )..Min( PicWidthInSamples_L, RefLayerPicWidthInSamples_L - xOffset )$$
$$\text{and } y = Max( 0, yOffset )..Min( PicHeightInSamples_L, RefLayerPicHeightInSamples_L - yOffset )$$

When ChromaArrayType is not equal to 0, for X being replaced by $rS_{Cb}$, $rS_{Cr}$, $cS_{Cb}$, and $cS_{Cr}$ and with currArray representing the array X of the collective term currentVars and refLayerArray representing the array X of the collective term refLayerVars, the array currArray is modified by

$$currArray[ x, y ] = refLayerArray[ x - xOffsetC, y - yOffsetC ] \quad (G\text{-}87)$$
$$\text{with } x = Max( 0, xOffsetC )..Min( PicWidthInSamples_C, RefLayerPicWidthInSamples_C - xOffsetC )$$
$$\text{and } y = Max( 0, yOffsetC )..Min( PicHeightInSamples_C, RefLayerPicHeightInSamples_C - yOffsetC )$$

### G.8.1.3 Layer representation decoding processes

Subclause G.8.1.3.1 specifies the base decoding process for layer representations without resolution change.

Subclause G.8.1.3.2 specifies the base decoding process for layer representations with resolution change.

Subclause G.8.1.3.3 specifies the target layer representation decoding process

#### G.8.1.3.1 Base decoding process for layer representations without resolution change

Inputs to this process are

- a variable currDQId specifying the current layer representation,
- a set of arrays collectively referred to as currentVars.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current layer representation with DQId equal to currDQId.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current layer representation with DQId equal to currDQId.

The base decoding process for layer representations without resolution change proceeds in the following ordered steps.

1. The variable noInterLayerPredFlag is set equal to the minimum value of no_inter_layer_pred_flag of the slices of the layer representation with DQId equal to currDQId.

2. Depending on noInterLayerPredFlag, the following applies.

   – If noInterLayerPredFlag is equal to 1, the array assignment and initialisation process as specified in subclause G.8.1.2.1 is invoked and the output is a modified set of arrays collectively referred to as currentVars.

   – Otherwise (noInterLayerPredFlag is equal to 0), the following ordered steps are specified.

      a. When tcoeff_level_prediction_flag is equal to 0, the macroblock address mbAddr proceeds over the values 0..(RefLayerPicSizeInMbs – 1), and for each macroblock address mbAddr, the macroblock decoding process prior to decoding a layer representation without resolution change and tcoeff_level_prediction_flag equal to 0 as specified in subclause G.8.1.5.4 is invoked with ref_layer_dq_id, mbAddr, and currentVars as the inputs and the output is a modified version of currentVars.

      b. When any of the variables ScaledRefLayerLeftOffset, ScaledRefLayerRightOffset, ScaledRefLayerTopOffset, or ScaledRefLayerBottomOffset is not equal to 0, the array restructuring process as specified in subclause G.8.1.2.2 is invoked with currentVars as the input and the output is a modified version of currentVars.

3. Let setOfSlices be the set of all slices of the current layer representation with DQId equal to currDQId. For each slice of the set setOfSlices, the base decoding process for slices without resolution change as specified in subclause G.8.1.4.1 is invoked with currSlice representing the currently processed slice, currDQId, and currentVars as the inputs and the output is a modified version of currentVars.

4. When currDQId is less than or equal to (DependencyIdMax << 4), with sliceIdc being the array sliceIdc of the collective term currentVars, the bitstream shall not contain data that results in any value of (sliceIdc[ mbAddr ] & 127) with mbAddr = 0..(PicSizeInMbs – 1) not equal to currDQId.

   NOTE – This constraint and a similar constraint in subclause G.8.1.3.2 specify that all layer representations with quality_id equal to 0 and all layer representations that are employed for inter-layer prediction shall be completely covered by the slices of the access unit. An additional constraint for layer representations with dependency_id equal to DependencyIdMax and quality_id greater than 0 is specified in subclause G.8.1.5.1.

**G.8.1.3.2 Base decoding process for layer representations with resolution change**

Inputs to this process are

– a variable currDQId specifying the current layer representation,

– a set of arrays collectively referred to as currentVars.

Outputs of this process are

– a set of arrays collectively referred to as refLayerVars,

– the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current layer representation with DQId equal to currDQId.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current layer representation with DQId equal to currDQId.

The base decoding process for layer representations with resolution change proceeds in the following ordered steps.

1. The macroblock address mbAddr proceeds over the values 0..(RefLayerPicSizeInMbs – 1), and for each macroblock address mbAddr, the macroblock decoding process prior to resolution change as specified in subclause G.8.1.5.5 is invoked with ref_layer_dq_id, mbAddr, and currentVars as the inputs and the output is a modified version of currentVars.

2. The deblocking filter process for Intra_Base prediction as specified in subclause G.8.7.1 is invoked with dependencyIdRefLayer set equal to (ref_layer_dq_id >> 4) and currentVars as the inputs and the output is a modified version of currentVars.

3. The variables that are assigned to the collective term currentVars are assigned to the collective term refLayerVars.

4. The array assignment and initialisation process as specified in subclause G.8.1.2.1 is invoked and the output is assigned to the collective term currentVars.

5. Let setOfSlices be the set of all slices of the current layer representation with DQId equal to currDQId. For each slice of the set setOfSlices, the base decoding process for slices with resolution change as specified in subclause G.8.1.4.2 is invoked with currSlice representing the currently processed slice, currDQId, refLayerVars, and currentVars as the inputs and the output is a modified version of currentVars.

6. With sliceIdc being the array sliceIdc of the collective term currentVars, the bitstream shall not contain data that results in any value of (sliceIdc[ mbAddr ] & 127) with mbAddr = 0..(PicSizeInMbs – 1) not equal to currDQId.

   NOTE – This constraint and a similar constraint in subclause G.8.1.3.1 specify that all layer representations with quality_id equal to 0 and all layer representation that are employed for inter-layer prediction shall be completely covered by the slices of the access unit. An additional constraint for layer representations with dependency_id equal to DependencyIdMax and quality_id greater than 0 is specified in subclause G.8.1.5.1.

**G.8.1.3.3 Target layer representation decoding process**

Inputs to this process are

– a variable currDQId specifying the current layer representation,

– a set of arrays collectively referred to as refLayerVars,

– a set of arrays collectively referred to as currentVars.

Outputs of this process are

– a (PicWidthInSamples$_L$)x(PicHeightInSamples$_L$) array $s_L$ containing constructed luma sample values,

– when ChromaArrayType is not equal to 0, two (PicWidthInSamples$_C$)x(PicHeightInSamples$_C$) arrays $s_{Cb}$ and $s_{Cr}$ containing constructed chroma sample values.

In this process the constructed samples of the array $s_L$ and, when ChromaArrayType is not equal to 0, the arrays $s_{Cb}$ and $s_{Cr}$ are derived using the variables that are assigned to currentVars.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current layer representation with DQId equal to currDQId.

The target layer representation decoding process proceeds in the following ordered steps.

1. The variables that are assigned to the collective term currentVars are assigned to the collective term tempVars, and in the following of this subclause, the arrays that are collectively referred to as tempVars are referred to by their names as specified in subclause G.8.1.2.1.

   NOTE 1 – Any following modification of the variables assigned to the collective term tempVars does not influence the variables assigned to the collective term currentVars.

2. The reference picture lists refPicList0 and refPicList1 are initially marked as not available.

3. The macroblock address mbAddr proceeds over the values 0..(PicSizeInMbs – 1), and for each macroblock address mbAddr, the following ordered steps are specified.

   a. Let currSlice specify the slice with DQId equal to ( ( ( sliceIdc[ mbAddr ] & 127 ) >> 4 ) << 4 ) and first_mb_in_slice equal to ( sliceIdc[ mbAddr ] >> 7 ).

   b. Let firstMbInSlice and sliceType be the syntax elements first_mb_in_slice and slice_type of the slice currSlice.

   c. When mbAddr is equal to firstMbInSlice, the reference picture lists refPicList0 and refPicList1 are marked as not available.

   d. When mbAddr is equal to firstMbInSlice and (sliceType % 5) is less than 2, the SVC decoding process for reference picture lists construction as specified in subclause G.8.2.3 is invoked with dependency_id, use_ref_base_pic_flag, and the current slice as the inputs and the outputs are the modified reference picture list refPicList0 and, when (sliceType % 5) is equal to 1, the modified reference picture list refPicList1.

      NOTE 2 – The reference picture lists refPicList0 and refPicList1 are only constructed for the slices of the layer representation with dependency_id equal to DependencyIdMax and quality_id equal to 0. For slices with dependency_id equal to DependencyIdMax and quality_id greater than 0, the reference picture lists are inferred.

e. The target macroblock decoding process as specified in subclause G.8.1.5.6 is invoked with currDQId, mbAddr, refLayerVars, tempVars, refPicList0 (when available), and refPicList1 (when available) as the inputs and the output is a modified version of tempVars.

NOTE 3 – Although the target layer representation decoding process is invoked twice for pictures with store_ref_base_pic_flag equal to 1, only a single motion compensation operation is needed for each macroblock.

4. The deblocking filter process for target representations as specified in subclause G.8.7.2 is invoked with tempVars as the input and the output is a modified version of tempVars.

5. All sample values of the array $S_L$ are copied to the array $s_L$, which is output of this subclause.

6. When ChromaArrayType is not equal to 0, all sample values of the arrays $S_{Cb}$ and $S_{Cr}$ are copied to the arrays $s_{Cb}$ and $s_{Cr}$, respectively, which are output of this subclause.

### G.8.1.4 Slice decoding processes

Subclause G.8.1.4.1 specifies the base decoding process for slices without resolution change.

Subclause G.8.1.4.2 specifies the base decoding process for slices with resolution change.

#### G.8.1.4.1 Base decoding process for slices without resolution change

Inputs to this process are

– the current slice currSlice,

– a variable currDQId specifying the current layer representation,

– a set of arrays collectively referred to as currentVars.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current slice currSlice.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the slice header of the current slice currSlice, the current picture parameter, which is identified by the syntax element pic_parameter_set_id inside the slice header of the current slice currSlice, and the current sequence parameter, which is identified by the syntax element seq_parameter_set_id inside the current picture parameter set.

When currDQId is equal to 0 and (slice_type % 5) is equal to 1, the SVC decoding process for reference picture lists construction as specified in subclause G.8.2.3 is invoked with currDependencyId equal to 0, useRefBasePicFlag equal to use_ref_base_pic_flag, and the current slice as input and the output is the reference picture list refPicList1.

The macroblocks of the current slice currSlice are processed in increasing order of their macroblock addresses. For each macroblock with macroblock address mbAddr, the base decoding process for macroblocks in slices without resolution change as specified in subclause G.8.1.5.2 is invoked with currDQId, mbAddr, currentVars, and, when currDQId is equal to 0 and (slice_type % 5) is equal to 1, the reference picture list refPicList1 as the inputs and the output is a modified version of currentVars.

#### G.8.1.4.2 Base decoding process for slices with resolution change

Inputs to this process are

– the current slice currSlice,

– a variable currDQId specifying the current layer representation,

– a set of arrays collectively referred to as refLayerVars,

– a set of arrays collectively referred to as currentVars.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current slice currSlice.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the slice header of the current slice currSlice, the current picture parameter, which is identified

by the syntax element pic_parameter_set_id inside the slice header of the current slice currSlice, and the current sequence parameter, which is identified by the syntax element seq_parameter_set_id inside the current picture parameter set.

When CroppingChangeFlag is equal to 1 and (slice_type % 5) is less than 2, the SVC decoding process for reference picture lists construction as specified in subclause G.8.2.3 is invoked with dependency_id, use_ref_base_pic_flag, and the current slice as the inputs and the outputs are the reference picture list refPicList0 and, when (sliceType % 5) is equal to 1, the reference picture list refPicList1.

The macroblocks of the current slice currSlice are processed in increasing order of their macroblock addresses. For each macroblock with macroblock address mbAddr, the base decoding process for macroblocks in slices with resolution change as specified in subclause G.8.1.5.3 is invoked with currDQId, mbAddr, refLayerVars, currentVars, refPicList0 (when CroppingChangeFlag is equal to 1 and (slice_type % 5) is less than 2), and refPicList1 (when CroppingChangeFlag is equal to 1 and (slice_type % 5) is equal to 1) as the inputs and the output is a modified version of currentVars.

### G.8.1.5 Macroblock initialisation and decoding processes

Subclause G.8.1.5.1 specifies the macroblock initialisation process.

Subclause G.8.1.5.2 specifies the base decoding process for macroblocks in slices without resolution change.

Subclause G.8.1.5.3 specifies the base decoding process for macroblocks in slices with resolution change.

Subclause G.8.1.5.4 specifies the macroblock decoding process prior to decoding a layer representation without resolution change and tcoeff_level_prediction_flag equal to 0.

Subclause G.8.1.5.5 specifies the macroblock decoding process prior to resolution change.

Subclause G.8.1.5.6 specifies the target macroblock decoding process.

#### G.8.1.5.1 Macroblock initialisation process

Inputs to this process are

- a set of arrays collectively referred to as refLayerVars,
- when CroppingChangeFlag is equal to 1 and (slice_type % 5) is less than 2, the reference picture list refPicList0,
- when CroppingChangeFlag is equal to 1 and (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Outputs of this process are

- a variable sliceIdc specifying the slice identification for the current macroblock,
- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a variable cTrafo specifying the transform type for the current macroblock,
- a variable mbType specifying the macroblock type of the current macroblock,
- a list subMbType with 4 elements specifying the sub-macroblock types of the current macroblock,
- a variable mvCnt specifying an initialisation value for the motion vector count of the current macroblock,
- a variable $tQP_Y$ specifying the luma quantisation parameter for the current macroblock,
- when ChromaArrayType is not equal to 0, two variables $tQP_{Cb}$ and $tQP_{Cr}$ specifying the chroma quantisation parameters for the current macroblock.
- two 2x2 arrays refIdxILPredL0 and refIdxILPredL1 specifying inter-layer predictors for the reference indices of the current macroblock,
- two 4x4x2 arrays mvILPredL0 and mvILPredL1 specifying inter-layer predictors for the motion vector components of the current macroblock.

Inside this subclause, the arrays sliceIdc, fieldMbFlag, cTrafo, mbType, subMbType, $tQP_Y$, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and sTCoeff that are collectively referred to as refLayerVars are referred to as refLayerSliceIdc, refLayerFieldMbFlag, refLayerCTrafo, refLayerMbType, refLayerSubMbType, $refLayerQP_Y$, refLayerPredFlagL0, refLayerPredFlagL1, refLayerRefIdxL0, refLayerRefIdxL1, refLayerMvL0, refLayerMvL1, and refLayerSTCoeff, respectively.

The variable sliceIdc is set equal to ( (first_mb_in_slice << 7 ) + DQId ).

The variable fieldMbFlag is derived as follows.

– If field_pic_flag is equal to 1, fieldMbFlag is set equal to 1.

– Otherwise, if SpatialResolutionChangeFlag is equal to 0 and slice_skip_flag is equal to 1, fieldMbFlag is set equal to refLayerFieldMbFlag[ CurrMbAddr ].

– Otherwise, fieldMbFlag is set equal to mb_field_decoding_flag.

The derivation process for macroblock type, sub-macroblock type, and inter-layer predictors for reference indices and motion vectors as specified in subclause G.8.1.5.1.1 is invoked with fieldMbFlag, refLayerFieldMbFlag, refLayerMbType, refLayerSubMbType, refLayerPredFlagL0, refLayerPredFlagL1, refLayerRefIdxL0, refLayerRefIdxL1, refLayerMvL0, refLayerMvL1, refPicList0 (when available), and refPicList1 (when available) as the inputs and the outputs are the variable mbType, the list subMbType, the 2x2 arrays refIdxILPredL0 and refIdxILPredL1, and the 4x4x2 arrays mvILPredL0 and mvILPredL1.

The derivation process for quantisation parameters and transform type as specified in subclause G.8.1.5.1.2 is invoked with mbType, subMbType, refLayerMbType, refLayerCTrafo, refLayer$QP_Y$, and refLayerSTCoeff as the inputs and the outputs are cTrafo, t$QP_Y$, and, when ChromaArrayType is not equal to 0, t$QP_{Cb}$ and t$QP_{Cr}$.

The variable mvCnt is set equal to 0.

When quality_id is greater than 0, the bitstream shall not contain data that result in (sliceIdcRefLayer[ CurrMbAddr ] & 127) not equal to (DQId – 1).

When no_inter_layer_pred_flag is equal to 0, SpatialResolutionChangeFlag is equal to 0, and fieldMbFlag is not equal to refLayerFieldMbRef[ CurrMbAddr ], the following constraints shall be obeyed.

a) The bitstream shall not contain data that result in base_mode_flag equal to 1, or any motion_prediction_flag_lX[ mbPartIdx ] with X being replaced by 0 and 1 and mbPartIdx = 0..3 equal to 1.

b) When residual_prediction_flag is equal to 1, the bitstream shall not contain data that result in any element refLayerSTCoeff[ CurrMbAddr ][ i ] not equal to 0 for i = 0..(255 + 2 * MbWidthC * MbHeightC).

**G.8.1.5.1.1 Derivation process for macroblock type, sub-macroblock type, and inter-layer predictors for reference indices and motion vectors**

Inputs to this process are

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying the macroblock types for the macroblocks of the reference layer representation,

– a (RefLayerPicSizeInMbs)x4 array refLayerSubMbType specifying the sub-macroblock types for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4 arrays refLayerPredFlagL0 and refLayerPredFlagL1 specifying prediction utilization flags for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4 arrays refLayerRefIdxL0 and refLayerRefIdxL1 specifying reference indices for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4x4x2 arrays refLayerMvL0 and refLayerMvL1 specifying motion vector components for the macroblocks of the reference layer representation,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is less than 2, the reference picture list refPicList0,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Outputs of this process are

– a variable mbType specifying the macroblock type of the current macroblock,

– a list subMbType with 4 elements specifying the sub-macroblock types of the current macroblock,

– two 2x2 arrays refIdxILPredL0 and refIdxILPredL1 specifying inter-layer predictors for the reference indices of the current macroblock,

– two 4x4x2 arrays mvILPredL0 and mvILPredL1 specifying inter-layer predictors for the motion vector components of the current macroblock.

The variable mbTypeILPred, the list subMbTypeILPred, the 2x2 arrays refIdxILPredL0 and refIdxILPredL1, and the 4x4x2 arrays mvILPredL0 and mvILPredL1 are derived as follows.

– If base_mode_flag is equal to 1 or any syntax element motion_prediction_flag_lX[ mbPartIdx ] with X being replaced by 0 and 1 and mbPartIdx = 0..3 is equal to 1, the derivation process for inter-layer predictors for macroblock type, sub-macroblock type, reference indices, and motion vectors as specified in subclause G.8.6.1 is invoked with fieldMbFlag, refLayerFieldMbFlag, refLayerMbType, refLayerSubMbType, refLayerPredFlagL0, refLayerPredFlagL1, refLayerRefIdxL0, refLayerRefIdxL1, refLayerMvL0, refLayerMvL1, refPicList0 (when available), and refPicList1 (when available) as the inputs and the outputs are the variable mbTypeILPred, the list subMbTypeILPred, the 2x2 arrays refIdxILPredL0 and refIdxILPredL1, and the 4x4x2 arrays mvILPredL0 and mvILPredL1.

– Otherwise (base_mode_flag is equal to 0 and all syntax elements motion_prediction_flag_lX[ mbPartIdx ] with X being replaced by 0 and 1 and mbPartIdx = 0..3 are equal to 0), mbTypeILPred is marked as not available, all elements subMbTypeILPred[ mbPartIdx ] with mbPartIdx = 0..3 of the list subMbTypeILPred are marked as not available, all elements of the 2x2 arrays refIdxILPredL0 and refIdxILPredL1 are set equal to –1, and all elements of the 4x4x2 arrays mvILPredL0 and mvILPredL1 are set equal to 0.

Depending on base_mode_flag, mb_type, and transform_size_8x8_flag, the variable mbType is derived as follows.

– If base_mode_flag is equal to 1, mbType is set equal to mbTypeILPred.

– Otherwise, if MbPartPredMode( mb_type, 0 ) is equal to Intra_4x4, mbType is set equal to I_4x4.

– Otherwise, if MbPartPredMode( mb_type, 0 ) is equal to Intra_8x8, mbType is set equal to I_8x8.

– Otherwise, if MbPartPredMode( mb_type, 0 ) is equal to Intra_16x16, mbType is set equal to I_16x16.

– Otherwise, if mb_type is equal to I_PCM, mbType is set equal to I_PCM.

– Otherwise (base_mode_flag is equal to 0 and mb_type specifies a P or B macroblock type), mbType is set equal to mb_type.

Depending on mbType and base_mode_flag, the list subMbType is derived as follows.

– If mbType is not equal to P_8x8 or B_8x8, all elements subMbType[ mbPartIdx ] with mbPartIdx = 0..3 are marked as unspecified.

– Otherwise, if base_mode_flag is equal to 1, each element subMbType[ mbPartIdx ] with mbPartIdx = 0..3 is set equal to subMbTypeILPred[ mbPartIdx ].

– Otherwise (mbType is equal to P_8x8 or B_8x8 and base_mode_flag is equal to 0), each element subMbType[ mbPartIdx ] with mbPartIdx = 0..3 is set equal to sub_mb_type[ mbPartIdx ].

When slice_type is equal to EP and base_mode_flag is equal to 1, the bitstream shall not contain data that result in any element refIdxILPredL0[ mbPartIdx ] with mbPartIdx = 0..3 that is less than 0.

When residual_prediction_flag equal to 1 is present in the bitstream, the bitstream shall not contain data that result in mbType being equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL.

#### G.8.1.5.1.2 Derivation process for quantisation parameters and transform type

Inputs to this process are

– a variable mbType specifying the macroblock type for the current macroblock,

– a list subMbType with 4 elements specifying the sub-macroblock types for the current macroblock,

– a one-dimensional array refLayerMbType specifying macroblock types for the macroblocks of the reference layer representation,

– a one-dimensional array refLayerCTrafo specifying transform types for the macroblocks of the reference layer representation,

– a one-dimensional array refLayer$QP_Y$ specifying luma quantisation parameters for the macroblocks of the reference layer representation,

– an (RefLayerPicSizeInMbs)x(256 + 2 * MbWidthC * MbHeightC) array refLayerSTCoeff specifying scaled transform coefficient values for the macroblocks of the reference layer representation.

Outputs of this process are

– a variable cTrafo specifying the transform type for the current macroblock,

– a variable $tQP_Y$ specifying the luma quantisation parameter for the current macroblock,

– when ChromaArrayType is not equal to 0, two variables $tQP_{Cb}$ and $tQP_{Cr}$ specifying the chroma quantisation parameters for the current macroblock.

The variable $tQP_Y$ is derived as follows.

– If SpatialResolutionChangeFlag is equal to 0, CodedBlockPatternLuma is equal to 0, CodedBlockPatternChroma is equal to 0, and any of the following conditions is true, $tQP_Y$ is set equal to refLayer$QP_Y$[ CurrMbAddr ].

  – mbType is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL and base_mode_flag is equal to 1

  – mbType is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL and residual_prediction_flag is equal to 1

– Otherwise, $tQP_Y$ is set equal to $QP_Y$.

When ChromaArrayType is not equal to 0, for CX being replaced by Cb and Cr, the variable $tQP_{CX}$ is set equal to the value of $QP_{CX}$ that corresponds to a value of $tQP_Y$ for $QP_Y$ as specified in subclause 8.5.8.

The variable predCoeffFlag is derived as follows.

– If SpatialResolutionChangeFlag is equal to 0 and any of the following conditions is true, predCoeffFlag is set equal to 1.

  – base_mode_flag is equal to 1, tcoeff_level_prediction_flag is equal to 0, and refLayerMbType[ CurrMbAddr ] is equal to I_BL,

  – base_mode_flag is equal to 1, tcoeff_level_prediction_flag is equal to 1, and mbType is equal to I_PCM, I_16x16, I_8x8, or I_4x4,

  – residual_prediction_flag is equal to 1 and mbType is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL

– Otherwise, predCoeffFlag is set equal to 0.

The variable cTrafo is derived as follows.

– If mbType is equal to I_PCM, cTrafo is set equal to T_PCM.

– Otherwise, if mbType is equal to I_16x16, cTrafo is set equal to T_16x16.

– Otherwise, if mbType is equal to I_8x8 or transform_size_8x8_flag is equal to 1, cTrafo is set equal to T_8x8.

– Otherwise, if predCoeffFlag is equal to 1, cTrafo is set equal to cTrafoRefLayer.

– Otherwise (predCoeffFlag is equal to 0, transform_size_8x8_flag is equal to 0, and mbType is not equal to I_PCM, I_16x16, or I_8x8), cTrafo is set equal to T_4x4.

When base_mode_flag is equal to 1, the following constraints shall be obeyed.

a) When mbType is equal to P_8x8 or B8x8 and NumSubMbPart( subMbType[ mbPartIdx ] ) is not equal to 1 for any mbPartIdx = 0..3, the bitstream shall not contain transform_size_8x8_flag equal to 1.

b) When mbType is equal to I_PCM, the bitstream shall not contain data that result in CodedBlockPatternLuma not equal to 0 or CodedBlockPatternChroma not equal to 0.

c) When mbType is equal to I_16x16 or I_4x4, the bitstream shall not contain transform_8x8_size_flag equal to 1.

d) When mbType is equal to I_8x8 and transform_size_8x8_flag is equal to 0, the bitstream shall not contain data that result in CodedBlockPatternLuma not equal to 0.

When predCoeffFlag is equal to 1, the following constraints shall be obeyed.

a) When refLayerCTrafo[ CurrMbAddr ] is equal to T_8x8 and transform_size_8x8_flag is equal 0, the bitstream shall not contain data that result in CodedBlockPatternLuma not equal to 0.

b) When refLayerCTrafo[ CurrMbAddr ] is equal to T_4x4 and transform_size_8x8_flag equal to 1, the bitstream shall not contain data that result in any element refLayerSTCoeff[ CurrMbAddr ][ i ] not equal to 0 for i = 0..(ChromaArrayType != 3 ? 255 : 767).

#### G.8.1.5.2 Base decoding process for macroblocks in slices without resolution change

Inputs to this process are

– a variable currDQId specifying the current layer representation,

– a variable mbAddr specifying the current macroblock inside the current layer representation,

– a set of arrays collectively referred to as currentVars,

– when currDQId is equal to 0 and (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current macroblock, which is specified by its address mbAddr and the layer representation identifier currDQId.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current macroblock, which is the macroblock with address mbAddr inside the layer representation with DQId equal to currDQId, the slice header of the current slice, which is the slice that contains the current macroblock, the current picture parameter, which is identified by the syntax element pic_parameter_set_id inside the slice header of the current slice, and the current sequence parameter, which is identified by the syntax element seq_parameter_set_id inside the current picture parameter set.

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

The base decoding process for macroblocks in slices without resolution change is specified by the following ordered steps.

1. The variable CurrMbAddr is set equal to mbAddr.
2. When tcoeff_level_prediction_flag is equal to 1, the variable $refQP_Y$ is set equal to $tQP_Y$[ mbAddr ] and, when ChromaArrayType is not equal to 0, the variables $refQP_{Cb}$ and $refQP_{Cr}$ are set equal to $tQP_{Cb}$[ mbAddr ] and $tQP_{Cr}$[ mbAddr ], respectively.
3. When no_inter_layer_pred_flag is equal to 0, the variable refLayerIntraBLFlag is derived as follows.
   – If mbType[ mbAddr ] is equal to I_BL, refLayerIntraBLFlag is set equal to 1.
   – Otherwise (mbType[ mbAddr ] is not equal to I_BL), refLayerIntraBLFlag is set equal to 0.
4. The macroblock initialisation process as specified in subclause G.8.1.5.1 is invoked with refLayerVars set equal to currentVars as the input and the outputs are assigned to sliceIdc[ mbAddr ], fieldMbFlag[ mbAddr ], cTrafo[ mbAddr ], mbType[ mbAddr ], subMbType[ mbAddr ], mvCnt[ mbAddr ], $tQP_Y$[ mbAddr ], $tQP_{Cb}$[ mbAddr ] (when ChromaArrayType is not equal to 0), $tQP_{Cr}$[ mbAddr ] (when ChromaArrayType is not equal to 0), the 2x2 arrays refIdxILPredL0 and refIdxILPredL1, and the 4x4x2 arrays mvILPredL0 and mvILPredL1.
5. The SVC derivation process for motion vector components and reference indices as specified in subclause G.8.4.1 is invoked with sliceIdc, fieldMbFlag, mbType, subMbType, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, mvCnt, refIdxILPredL0, refIdxILPredL1, mvILPredL0, mvILPredL1, and refPicList1 (when available) as the inputs and the outputs are modified versions of the arrays predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and mvCnt.
6. Depending on mbType[ mbAddr ], the following applies.
   – If mbType[ mbAddr ] is equal to I_PCM, I_16x16, I_8x8, or I_4x4, the following ordered steps are specified.
     a. When base_mode_flag is equal to 0, the SVC derivation process for intra prediction modes as specified in subclause G.8.3.1 is invoked with sliceIdc, fieldMbFlag, mbType, ipred4x4, ipred8x8, ipred16x16, and, when ChromaArrayType is equal to 1 or 2, ipredChroma as the inputs and the outputs are modified versions of ipred4x4, ipred8x8, ipred16x16, and, when ChromaArrayType is equal to 1 or 2, ipredChroma.
     b. When base_mode_flag is equal to 1, the transform coefficient level scaling process prior to transform coefficient refinement as specified in subclause G.8.5.2 is invoked with cTrafo[ mbAddr ], tCoeffLevel[ mbAddr ], $tQP_Y$[ mbAddr ], $refQP_Y$, and, when ChromaArrayType is not equal to 0,

$tQP_{Cb}$[ mbAddr ], $tQP_{Cr}$[ mbAddr ], $refQP_{Cb}$, and $refQP_{Cr}$ as the inputs and the output is a modified version of tCoeffLevel[ mbAddr ].

c. The transform coefficient scaling and refinement process as specified in subclause G.8.5.1 is invoked with refinementFlag set equal to base_mode_flag, fieldMbFlag[ mbAddr ], cTrafo[ mbAddr ], sTCoeff[ mbAddr ], and tCoeffLevel[ mbAddr ] as the inputs and the outputs are modified versions of sTCoeff[ mbAddr ] and tCoeffLevel[ mbAddr ].

d. The sample array re-initialisation process as specified in subclause G.8.5.5 is invoked with fieldMbFlag[ mbAddr ], $rS_L$, and, when ChromaArrayType is not equal to 0, $rS_{Cb}$ and $rS_{Cr}$ as the inputs and the outputs are a modified version of $rS_L$ and, when ChromaArrayType is not equal to 0, modified versions of $rS_{Cb}$ and $rS_{Cr}$.

e. The sample array re-initialisation process as specified in subclause G.8.5.5 is invoked with fieldMbFlag[ mbAddr ], $cS_L$, and, when ChromaArrayType is not equal to 0, $cS_{Cb}$ and $cS_{Cr}$ as the inputs and the outputs are a modified version of $cS_L$ and, when ChromaArrayType is not equal to 0, modified versions of $cS_{Cb}$ and $cS_{Cr}$.

– Otherwise, if mbType[ mbAddr ] is equal to I_BL, the transform coefficient scaling and refinement process as specified in subclause G.8.5.1 is invoked with refinementFlag set equal to refLayerIntraBLFlag, fieldMbFlag[ mbAddr ], cTrafo[ mbAddr ], sTCoeff[ mbAddr ], and tCoeffLevel[ mbAddr ] as the inputs and the outputs are modified versions of sTCoeff[ mbAddr ] and tCoeffLevel[ mbAddr ].

– Otherwise (mbType[ mbAddr ] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL), the following ordered steps are specified.

a. When residual_prediction_flag is equal to 1, the transform coefficient level scaling process prior to transform coefficient refinement as specified in subclause G.8.5.2 is invoked with cTrafo[ mbAddr ], tCoeffLevel[ mbAddr ], $tQP_Y$[ mbAddr ], $refQP_Y$, and, when ChromaArrayType is not equal to 0, $tQP_{Cb}$[ mbAddr ], $tQP_{Cr}$[ mbAddr ], $refQP_{Cb}$, and $refQP_{Cr}$ as the inputs and the output is a modified version of tCoeffLevel[ mbAddr ].

b. The transform coefficient scaling and refinement process as specified in subclause G.8.5.1 is invoked with refinementFlag set equal to residual_prediction_flag, fieldMbFlag[ mbAddr ], cTrafo[ mbAddr ], sTCoeff[ mbAddr ], and tCoeffLevel[ mbAddr ] as the inputs and the outputs are modified versions of sTCoeff[ mbAddr ] and tCoeffLevel[ mbAddr ].

c. When residual_prediction_flag is equal to 0, the sample array re-initialisation process as specified in subclause G.8.5.5 is invoked with fieldMbFlag[ mbAddr ], $rS_L$, and, when ChromaArrayType is not equal to 0, $rS_{Cb}$ and $rS_{Cr}$ as the inputs and the outputs are a modified versions of $rS_L$ and, when ChromaArrayType is not equal to 0, modified versions of $rS_{Cb}$ and $rS_{Cr}$.

d. When residual_prediction_flag is equal to 0, the sample array re-initialisation process as specified in subclause G.8.5.5 is invoked with fieldMbFlag[ mbAddr ], $cS_L$, and, when ChromaArrayType is not equal to 0, $cS_{Cb}$ and $cS_{Cr}$ as the inputs and the outputs are a modified versions of $cS_L$ and, when ChromaArrayType is not equal to 0, modified versions of $cS_{Cb}$ and $cS_{Cr}$.

**G.8.1.5.3 Base decoding process for macroblocks in slices with resolution change**

Inputs to this process are

– a variable currDQId specifying the current layer representation,

– a variable mbAddr specifying the current macroblock inside the current layer representation,

– a set of arrays collectively referred to as refLayerVars,

– a set of arrays collectively referred to as currentVars,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is less than 2, the reference picture list refPicList0,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current macroblock, which is specified by its address mbAddr and the layer representation identifier currDQId, as well as variables assigned to refLayerVars.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current macroblock, which is the macroblock with address mbAddr inside the layer representation with DQId equal to currDQId, the slice header of the current slice, which is the slice that contains the current macroblock, the current picture parameter, which is identified by the syntax element pic_parameter_set_id inside the slice header of the current slice, and the current sequence parameter, which is identified by the syntax element seq_parameter_set_id inside the current picture parameter set.

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

Inside this subclause, the arrays sliceIdc, fieldMbFlag, cTrafo, mbType, $cS_L$, $cS_{Cb}$, $cS_{Cr}$, $rS_L$, $rS_{Cb}$, and $rS_{Cr}$ of the collective term refLayerVars are referred to as refLayerSliceIdc, refLayerFieldMbFlag, refLayerCTrafo, refLayerMbType, $refS_L$, $refS_{Cb}$, $refS_{Cr}$, $refR_L$, $refR_{Cb}$, and $refR_{Cr}$, respectively.

The base decoding process for macroblocks in slices with resolution change is specified by the following ordered steps.

1. The variable CurrMbAddr is set equal to mbAddr.
2. The macroblock initialisation process as specified in subclause G.8.1.5.1 is invoked with refLayerVars, refPicList0 (when available), and refPicList1 (when available) as the inputs and the outputs are assigned to sliceIdc[ mbAddr ], fieldMbFlag[ mbAddr ], cTrafo[ mbAddr ], mbType[ mbAddr ], subMbType[ mbAddr ], mvCnt[ mbAddr ], $tQP_Y$[ mbAddr ], $tQP_{Cb}$[ mbAddr ] (when ChromaArrayType is not equal to 0), $tQP_{Cr}$[ mbAddr ] (when ChromaArrayType is not equal to 0), the 2x2 arrays refIdxILPredL0 and refIdxILPredL1, and the 4x4x2 arrays mvILPredL0 and mvILPredL1.
3. The SVC derivation process for motion vector components and reference indices as specified in subclause G.8.4.1 is invoked with sliceIdc, fieldMbFlag, mbType, subMbType, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, mvCnt, refIdxILPredL0, refIdxILPredL1, mvILPredL0, and mvILPredL1 as the inputs and the outputs are modified versions of the arrays predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and mvCnt.
4. The variable intraResamplingFlag is derived as follows.

   If any of the following conditions is true, intraResamplingFlag is set equal to 1.

   – RestrictedSpatialResolutionChangeFlag is equal to 1 and mbType[ mbAddr ] is equal to I_BL

   – RestrictedSpatialResolutionChangeFlag is equal to 0, MbaffFrameFlag is equal to 0, RefLayerMbaffFrameFlag is equal to 0, and mbType[ mbAddr ] is not equal to I_PCM, I_16x16, I_8x8, or I_4x4

   – Otherwise, intraResamplingFlag is set equal to 0.
5. When intraResamplingFlag is equal to 1, the resampling process for intra samples as specified in subclause G.8.6.2 is invoked with fieldMbFlag[ mbAddr ], refLayerSliceIdc, refLayerFieldMbFlag, refLayerMbType, $refS_L$, $cS_L$, and, when ChromaArrayType is not equal to 0, $refS_{Cb}$, $refS_{Cr}$, $cS_{Cb}$, and $cS_{Cr}$ as the inputs and the outputs are a modified version of the array $cS_L$ and, when ChromaArrayType is not equal to 0, modified versions of the array $cS_{Cb}$, and $cS_{Cr}$.
6. Depending on mbType[ mbAddr ], the following applies.

   – If mbType[ mbAddr ] is equal to I_PCM, I_16x16, I_8x8, or I_4x4, the SVC derivation process for intra prediction modes as specified in subclause G.8.3.1 is invoked with sliceIdc, fieldMbFlag, mbType, ipred4x4, ipred8x8, ipred16x16, and, when ChromaArrayType is equal to 1 or 2, ipredChroma as the inputs and the outputs are modified versions of ipred4x4, ipred8x8, ipred16x16, and, when ChromaArrayType is equal to 1 or 2, ipredChroma.

   – Otherwise, if mbType[ mbAddr ] is not equal to I_BL and residual_prediction_flag is equal to 1, the resampling process for residual samples as specified in subclause G.8.6.3 is invoked with fieldMbFlag[ mbAddr ], refLayerFieldMbFlag, refLayerCTrafo, $refR_L$, $rS_L$, and, when ChromaArrayType is not equal to 0, $refR_{Cb}$, $refR_{Cr}$, $rS_{Cb}$, and $rS_{Cr}$ as the inputs and the outputs are a modified version of the array $rS_L$ and, when ChromaArrayType is not equal to 0, modified versions of the array $rS_{Cb}$, and $rS_{Cr}$.

   – Otherwise, the arrays of the collective term currentVars are not modified.
7. The transform coefficient scaling and refinement process as specified in subclause G.8.5.1 is invoked with refinementFlag set equal to 0, fieldMbFlag[ mbAddr ], cTrafo[ mbAddr ], sTCoeff[ mbAddr ], and tCoeffLevel[ mbAddr ] as the inputs and the outputs are modified versions of sTCoeff[ mbAddr ] and tCoeffLevel[ mbAddr ].

#### G.8.1.5.4 Macroblock decoding process prior to decoding a layer representation without resolution change and tcoeff_level_prediction_flag equal to 0

Inputs to this process are

– a variable currDQId specifying the current layer representation,

– a variable mbAddr specifying the current macroblock inside the current layer representation,

– a set of arrays collectively referred to as currentVars.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current macroblock, which is specified by its address mbAddr and the layer representation identifier currDQId.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current macroblock, which is the macroblock with address mbAddr inside the layer representation with DQId equal to currDQId, the slice header of the current slice, which is the slice that contains the current macroblock, the current picture parameter, which is identified by the syntax element pic_parameter_set_id inside the slice header of the current slice, and the current sequence parameter, which is identified by the syntax element seq_parameter_set_id inside the current picture parameter set.

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

The target macroblock decoding process is specified by the following ordered steps.

1. The variable CurrMbAddr is set equal to mbAddr.
2. When mbType[ mbAddr ] is equal to I_PCM, I_16x16, I_8x8, or I_4x4, the SVC intra sample prediction and construction process as specified in subclause G.8.3.2 is invoked with sliceIdc, fieldMbFlag, mbType, ipred4x4[ mbAddr ], ipred8x8[ mbAddr ], ipred16x16[ mbAddr ], ipredChroma[ mbAddr ], cTrafo[ mbAddr ], sTCoeff[ mbAddr ], $cS_L$, and, when ChromaArrayType is not equal to 0, $cS_{Cb}$ and $cS_{Cr}$ as the inputs and the outputs are a modified version of the array $cS_L$ and, when ChromaArrayType is not equal to 0, modified versions of the arrays $cS_{Cb}$ and $cS_{Cr}$.

#### G.8.1.5.5 Macroblock decoding process prior to resolution change

Inputs to this process are

– a variable currDQId specifying the current layer representation,

– a variable mbAddr specifying the current macroblock inside the current layer representation,

– a set of arrays collectively referred to as currentVars.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current macroblock, which is specified by its address mbAddr and the layer representation identifier currDQId.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current macroblock, which is the macroblock with address mbAddr inside the layer representation with DQId equal to currDQId, the slice header of the current slice, which is the slice that contains the current macroblock, the current picture parameter, which is identified by the syntax element pic_parameter_set_id inside the slice header of the current slice, and the current sequence parameter, which is identified by the syntax element seq_parameter_set_id inside the current picture parameter set.

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

The target macroblock decoding process is specified by the following ordered steps.

1. The variable CurrMbAddr is set equal to mbAddr.
2. Depending on mbType[ mbAddr ], the following applies.
   – If mbType[ mbAddr ] is equal to I_PCM, I_16x16, I_8x8, or I_4x4, the SVC intra sample prediction and construction process as specified in subclause G.8.3.2 in invoked with sliceIdc, fieldMbFlag, mbType,

ipred4x4[ mbAddr ], ipred8x8[ mbAddr ], ipred16x16[ mbAddr ], ipredChroma[ mbAddr ], cTrafo[ mbAddr ], sTCoeff[ mbAddr ], $cS_L$, and, when ChromaArrayType is not equal to 0, $cS_{Cb}$ and $cS_{Cr}$ as the inputs and the outputs are a modified version of the array $cS_L$ and, when ChromaArrayType is not equal to 0, modified versions of the arrays $cS_{Cb}$ and $cS_{Cr}$.

– Otherwise, if mbType[ mbAddr ] is equal to I_BL, the following ordered steps are specified.

  a. The residual construction and accumulation process as specified in subclause G.8.5.3 is invoked with accumulationFlag set equal to 0, fieldMbFlag[ mbAddr ], cTrafo[ mbAddr ], sTCoeff[ mbAddr ], $rS_L$, and, when ChromaArrayType is not equal to 0, $rS_{Cb}$ and $rS_{Cr}$ as the inputs and the outputs are a modified version of $rS_L$ and, when ChromaArrayType is not equal to 0, modified versions of $rS_{Cb}$ and $rS_{Cr}$.

  b. The sample array accumulation process as specified in subclause G.8.5.4 is invoked with fieldMbFlag[ mbAddr ], $rS_L$, $cS_L$, and, when ChromaArrayType is not equal to 0, $rS_{Cb}$, $rS_{Cr}$ $cS_{Cb}$, and $cS_{Cr}$ as the inputs and the outputs are a modified version of $cS_L$ and, when ChromaArrayType is not equal to 0, modified versions of $cS_{Cb}$ and $cS_{Cr}$.

– Otherwise (mbType[ mbAddr ] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL), the following ordered steps are specified.

  a. The residual construction and accumulation process as specified in subclause G.8.5.3 is invoked with accumulationFlag set equal to 1, fieldMbFlag[ mbAddr ], cTrafo[ mbAddr ], sTCoeff[ mbAddr ], $rS_L$, and, when ChromaArrayType is not equal to 0, $rS_{Cb}$ and $rS_{Cr}$ as the inputs and the outputs are a modified version of $rS_L$ and, when ChromaArrayType is not equal to 0, modified versions of $rS_{Cb}$ and $rS_{Cr}$.

  b. The sample array re-initialisation process as specified in subclause G.8.5.5 is invoked with fieldMbFlag[ mbAddr ], $cS_L$, and, when ChromaArrayType is not equal to 0, $cS_{Cb}$ and $cS_{Cr}$ as the inputs and the outputs are a modified versions of $cS_L$ and, when ChromaArrayType is not equal to 0, modified versions of $cS_{Cb}$ and $cS_{Cr}$.

#### G.8.1.5.6 Target macroblock decoding process

Inputs to this process are

– a variable currDQId specifying the current layer representation,

– a variable mbAddr specifying the current macroblock inside the current layer representation,

– a set of arrays collectively referred to as refLayerVars,

– a set of arrays collectively referred to as currentVars,

– when (slice_type % 5) is less than 2, the reference picture list refPicList0,

– when (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and derived upper-case variables for the current macroblock, which is specified by its address mbAddr and the layer representation identifier currDQId.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the current macroblock, which is the macroblock with address mbAddr inside the layer representation with DQId equal to currDQId, the slice header of the current slice, which is the slice that contains the current macroblock, the current picture parameter, which is identified by the syntax element pic_parameter_set_id inside the slice header of the current slice, and the current sequence parameter, which is identified by the syntax element seq_parameter_set_id inside the current picture parameter set.

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

Inside this subclause, the arrays fieldMbFlag and mbType of the collective term refLayerVars are referred to as refLayerFieldMbFlag and refLayerMbType, respectively.

The target macroblock decoding process is specified by the following ordered steps.

1. The variable CurrMbAddr is set equal to mbAddr.

2. When tcoeff_level_prediction is equal to 1, (sliceIdc[ mbAddr ] &127 ) is not equal to currDQId, and ChromaArrayType is not equal to 0, the following ordered steps are specified.

   NOTE – By the following ordered steps, only the transform coefficient levels for chroma are scaled (and the scaled transform coefficients are derived accordingly) when the chroma quantisation parameter offsets of the current layer representation with DQId equal to currDQId and the layer representation with DQId equal to (sliceIdc[ mbAddr ] &127 ) are different.

   a. The variable $cQP_Y$ is set equal to $tQP_Y$[ mbAddr ], and for CX being replaced by Cb and Cr, the variable $cQP_{CX}$ is set equal to the value of $QP_{CX}$ that corresponds to a value of $cQP_Y$ for $QP_Y$ as specified in subclause 8.5.8.

   b. The transform coefficient level scaling process prior to transform coefficient refinement as specified in subclause G.8.5.2 is invoked with cTrafo[ mbAddr ], tCoeffLevel[ mbAddr ], $tQP_Y$ set equal to $cQP_Y$, $refQP_Y$ set equal to $tQP_Y$[ mbAddr ], and, when ChromaArrayType is not equal to 0, $tQP_{Cb}$ set equal to $cQP_{Cb}$, $tQP_{Cr}$ set equal to $cQP_{Cr}$, $refQP_{Cb}$ set equal to $tQP_{Cb}$[ mbAddr ], $refQP_{Cr}$ set equal to $tQP_{Cr}$[ mbAddr ] as the inputs and the output is a modified version of tCoeffLevel[ mbAddr ].

   c. The transform coefficient scaling and refinement process as specified in subclause G.8.5.1 is invoked with refinementFlag equal to 1, fieldMbFlag[ mbAddr ], cTrafo[ mbAddr ], sTCoeff[ mbAddr ], and tCoeffLevel[ mbAddr ] as the inputs and the outputs are modified versions of sTCoeff[ mbAddr ] and tCoeffLevel[ mbAddr ]. For this invocation of the process in subclause G.8.5.1, all elements of the arrays LumaLevel, LumaLevel8x8, Intra16x16DCLevel, Intra16x16ACLevel, CbLevel, CbLevel8x8, CbIntra16x16DCLevel, CbIntra16x16ACLevel, CrLevel, CrLevel8x8, CrIntra16x16DCLevel, CrIntra16x16ACLevel, ChromaDCLevel, and ChromaACLevel are inferred to be equal to 0, and the quantisation parameter $QP'_Y$ is set equal to ($tQP_Y$[ mbAddr ] + $QpBdOffset_Y$).

3. Depending on mbType[ mbAddr ], the following applies.

   – If mbType[ mbAddr ] is equal to I_PCM, I_16x16, I_8x8, or I_4x4, the SVC intra sample prediction and construction process as specified in subclause G.8.3.2 in invoked with sliceIdc, fieldMbFlag, mbType, ipred4x4[ mbAddr ], ipred8x8[ mbAddr ], ipred16x16[ mbAddr ], ipredChroma[ mbAddr ], cTrafo[ mbAddr ], sTCoeff[ mbAddr ], $cS_L$, and, when ChromaArrayType is not equal to 0, $cS_{Cb}$ and $cS_{Cr}$ as the inputs and the outputs are a modified version of the array $cS_L$ and, when ChromaArrayType is not equal to 0, modified versions of the arrays $cS_{Cb}$ and $cS_{Cr}$.

   – Otherwise, if mbType[ mbAddr ] is equal to I_BL, the following ordered steps are specified.

     a. The residual construction and accumulation process as specified in subclause G.8.5.3 is invoked with accumulationFlag set equal to 0, fieldMbFlag[ mbAddr ], cTrafo[ mbAddr ], sTCoeff[ mbAddr ], $rS_L$, and, when ChromaArrayType is not equal to 0, $rS_{Cb}$ and $rS_{Cr}$ as the inputs and the outputs are a modified version of $rS_L$ and, when ChromaArrayType is not equal to 0, modified versions of $rS_{Cb}$ and $rS_{Cr}$.

     b. The sample array accumulation process as specified in subclause G.8.5.4 is invoked with fieldMbFlag[ mbAddr ], $rS_L$, $cS_L$, and, when ChromaArrayType is not equal to 0, $rS_{Cb}$, $rS_{Cr}$ $cS_{Cb}$, and $cS_{Cr}$ as the inputs and the outputs are a modified version of $cS_L$ and, when ChromaArrayType is not equal to 0, modified versions of $cS_{Cb}$ and $cS_{Cr}$.

   – Otherwise (mbType[ mbAddr ] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL), the following ordered steps are specified.

     a. The residual construction and accumulation process as specified in subclause G.8.5.3 is invoked with accumulationFlag set equal to 1, fieldMbFlag[ mbAddr ], cTrafo[ mbAddr ], sTCoeff[ mbAddr ], $rS_L$, and, when ChromaArrayType is not equal to 0, $rS_{Cb}$ and $rS_{Cr}$ as the inputs and the outputs are a modified version of $rS_L$ and, when ChromaArrayType is not equal to 0, modified versions of $rS_{Cb}$ and $rS_{Cr}$.

     b. The SVC decoding process for Inter prediction samples as specified in subclause G.8.4.2 is invoked with fieldMbFlag[ mbAddr ], mbType[ mbAddr ], subMbType[ mbAddr ], predFlagL0[ mbAddr ], predFlagL1[ mbAddr ], refIdxL0[ mbAddr ], refIdxL1[ mbAddr ], mvL0[ mbAddr ], mvL1[ mbAddr ], refLayerFieldMbFlag, refLayerMbType, refPicList0, refPicList1 (when available), $cS_L$, and, when ChromaArrayType is not equal to 0, $cS_{Cb}$ and $cS_{Cr}$ as the inputs and the outputs are a modified version of $cS_L$ and, when ChromaArrayType is not equal to 0, modified versions of $cS_{Cb}$ and $cS_{Cr}$.

     c. The sample array accumulation process as specified in subclause G.8.5.4 is invoked with fieldMbFlag[ mbAddr ], $rS_L$, $cS_L$, and, when ChromaArrayType is not equal to 0, $rS_{Cb}$, $rS_{Cr}$ $cS_{Cb}$, and

$cS_{Cr}$ as the inputs and the outputs are a modified version of $cS_L$ and, when ChromaArrayType is not equal to 0, modified versions of $cS_{Cb}$ and $cS_{Cr}$.

4. The variable MvCnt for the macroblock mbAddr is set equal to mvCnt[ mbAddr ].

### G.8.2 SVC reference picture lists construction and decoded reference picture marking process

The SVC decoding process for picture order count is specified in subclause G.8.2.1.

The SVC decoding process for picture numbers is specified in subclause G.8.2.2.

The SVC decoding process for reference picture lists construction is specified in subclause G.8.2.3.

The SVC decoded reference picture marking process is specified in subclause G.8.2.4.

The SVC decoding process for gaps in frame_num is specified in subclause G.8.2.5.

The decoding process for picture order counts is independently applied for different values of dependency_id. Syntax elements that are related to picture order count for a particular value of dependency_id do not influence the derivation of picture order counts for other values of dependency_id.

The reference picture marking is independently applied for different values of dependency_id. Syntax elements that are related to reference picture marking for a particular value of dependency_id do not influence the reference picture marking for other values of dependency_id.

The decoding process for gaps is frame_num is independently applied for different values of dependency_id.

Reference picture lists for different dependency representations are constructed independently. Syntax elements that are related to reference picture list construction for a particular value of dependency_id do not influence the reference picture list construction for other values of dependency_id. Reference picture list for a particular value of dependency_id are constructed based on the reference picture marking for this particular value of dependency_id. The reference picture marking for a particular value of dependency_id does not influence the reference picture list construction for a different value of dependency_id.

Only the elements of the reference picture lists for dependency_id equal to DependencyIdMax represent decoded pictures that are associated with decoded samples. Only the reference picture lists for dependency_id equal to DependencyIdMax are used for the derivation of inter prediction samples as specified in subclause G.8.4.2. The elements of the reference picture lists for dependency representation with dependency_id less than DependencyIdMax represent layer pictures, which are not associated with decoded samples. The elements of the reference picture lists for dependency_id equal to 0 are associated with the arrays fieldMbFlag, mbType, subMbType, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, and mvL1 as specified in subclause G.8.1.2.1 that were derived when decoding the layer representation with dependency_id equal to 0 and quality_id equal to 0 for the corresponding access unit. These arrays are used for the derivation of motion vectors and reference indices for layer representation with dependency_id equal to 0 and quality_id equal to 0 as specified in subclause G.8.4.1.2. The elements of the reference picture lists for all dependency representations with dependency_id greater than 0 are associated with the variables ScaledRefLayerLeftOffset, ScaledRefLayerRightOffset, ScaledRefLayerTopOffset, and ScaledRefLayerBottomOffset. These variables are used for deriving inter-layer motion vector predictions as specified in subclause G.8.6.1.2.

NOTE – For each access unit, decoded samples only need to be stored for the dependency representation with dependency_id equal to DependencyIdMax, and motion data arrays only need to be stored for the dependency representation with dependency_id equal to 0.

The SVC decoding processes for picture order count, reference picture list construction, reference picture marking, and gaps in frame_num are specified using processes specified in subclause 8. The following modifications to the processes specified in this subclause and the processes of subclause 8 that are invoked from these processes apply with currDependencyId representing the value of dependency_id for the dependency representation for which the processes are invoked.

a) All syntax elements and derived upper-case variables that are referred to in this process or in a child process invoked from this process are syntax elements and upper-case variables for the dependency representation with dependency_id equal to currDependencyId.

b) When dependency_id is less than DependencyIdMax, the following applies.

   – A frame, field, top field, bottom field, picture, and decoded picture is interpreted as layer frame, layer field, layer top field, layer bottom field, layer picture, and decoded layer picture, respectively, that represent an intermediate decoding result for the dependency representation with dependency_id equal to currDependencyId.

   – An IDR picture is interpreted as layer picture with IdrPicFlag is equal to 1 for the dependency representation with dependency_id equal to currDependencyId.

– A reference frame, reference field, and reference picture is interpreted as layer frame, layer field, and layer picture with nal_ref_idc greater than 0 for the dependency representation with dependency_id equal to currDependencyId.

– A non-reference frame, non-reference field, and non-reference picture is interpreted as layer frame, layer field, and layer picture with nal_ref_idc equal to 0 for the dependency representation with dependency_id equal to currDependencyId.

– A complementary non-reference field pair is interpreted as complementary non-reference layer field pair for the dependency representation with dependency_id equal to currDependencyId. A complementary non-reference layer field pair for a particular value of dependency_id is a pair of two layer fields with nal_ref_idc equal to 0 for the particular value of dependency_id that are in consecutive access units containing a dependency representation with the particular value of dependency_id as two coded layer field and have opposite parity where the first layer field is not already a paired layer field.

– A complementary reference field pair is interpreted as complementary reference layer field pair for the dependency representation with dependency_id equal to currDependencyId. A complementary reference layer field pair for a particular value of dependency_id is a pair of two layer fields with nal_ref_idc greater than 0 for the particular value of dependency_id that are in consecutive access units containing a dependency representation with the particular value of dependency_id as two coded layer fields and share the same value of frame_num where the second layer field in decoding order has IdrPicFlag equal to 0 and does not include a memory_management_control_operation syntax element equal to 5.

– A complementary field pair is interpreted as complementary layer field pair for the dependency representation with dependency_id equal to currDependencyId. A complementary layer field pair is a collective term for a complementary reference layer field pair and a complementary non-reference layer field pair.

– A non-paired non-reference field is interpreted as layer field with nal_ref_idc equal to 0 for the dependency representation with dependency_id equal to currDependencyId that is not part of a complementary non-reference layer field pair.

– A non-paired reference field is interpreted as layer field with nal_ref_idc greater than 0 for the dependency representation with dependency_id equal to currDependencyId that is not part of a complementary reference layer field pair.

– A non-paired reference field is interpreted as layer field for the dependency representation with dependency_id equal to currDependencyId that is not part of a complementary layer field pair.

#### G.8.2.1 SVC decoding process for picture order count

Outputs of this process are the variables TopFieldOrderCnt (if applicable) and BottomFieldOrderCnt (if applicable) for all dependency representations of the current access unit.

For all dependency representations of the current access unit, the variables TopFieldOrderCnt (if applicable) and BottomFieldOrderCnt (if applicable) are derived by invoking the decoding process for picture order count as specified in subclause 8.2.1. For these invocations of the process specified in subclause 8.2.1, the modifications a) and b) specified in subclause G.8.2 apply with currDependencyId being equal to dependency_id of the corresponding dependency representation.

For all dependency representations of an access unit, either TopFieldOrderCnt or BottomFieldOrderCnt or both are derived. When both are derived in two or more dependency representations of an access unit, their difference shall be the same in these dependency representations of the access unit.

The values of TopFieldOrderCount and BottomFieldOrderCount are restricted as follows.

– For each access unit, the one-dimensional array picOrderCnt is derived as follows.

  – If TopFieldOrderCnt is derived in all dependency representations of an access unit, the variable picOrderCnt[ dId ] is set equal to TopFieldOrderCnt of the dependency representation with dependency_id being equal to dId.

  – Otherwise (TopFieldOrderCnt is not derived in all dependency representations of an access unit), the variable picOrderCnt[ dId ] is set equal to BottomFieldOrderCnt of the dependency representation with dependency_id being equal to dId.

– Let au0 and au1 be any pair of access units in the bitstream with au1 being later in decoding order than au0.

– Let the flag idrConditionFlag be derived for each dependency representation in an access unit as follows.

- If the dependency representation in the access unit has IdrPicFlag equal to 1 or a memory_management_control_operation syntax element equal to 5, idrConditionFlag is set equal to 1.
- Otherwise (the dependency representation in the access unit has IdrPicFlag equal to 0 and does not have a memory_management_control_operation syntax element equal to 5), idrConditionFlag is set equal to 0.

- Let the set dIdSet0 be the set of all dependency_id values in au0.
- Let the set dIdSet1 be the set of all dependency_id values in au1 for which idrConditionFlag is not equal to 1 in any access unit in decoding order between the access unit that follows au0 and the access unit au1, inclusive.
- For all values of dId that are present in both sets dIdSet0 and dIdSet1, the differences between the value picOrderCnt[ dId ] in au0 and the value picOrderCnt[ dId ] in au1 shall be the same.

#### G.8.2.2 SVC decoding process for picture numbers

This process is invoked when the SVC decoding process for reference picture lists construction specified in subclause G.8.2.3 is invoked or when the SVC reference picture marking process for a dependency representation as specified in subclause G.8.2.4.1 is invoked.

Inputs to this process are

- a variable currDependencyId specifying a dependency representation,
- a variable useRefBasePicFlag specifying whether this process refers to reference base pictures or decoded pictures.

Outputs of this process are variables FrameNum, FrameNumWrap, PicNum, LongTermFrameIdx, and LongTermPicNum assigned to all pictures marked as "used for short-term reference" or "used for long-term reference".

From here to the end of this subclause, the modifications a) and b) specified in subclause G.8.2 apply.

The variable maxFrameNum is derived by

$$maxFrameNum = ( 1 << ( log2\_max\_frame\_num\_minus4 + 4 ) ) \quad \text{(G-88)}$$

To each short-term and long-term reference picture a variable useFlag is assigned as follows.

- If useRefBasePicFlag is equal to 0, the following applies.
  - If the reference picture is not marked as "reference base picture", useFlag is set equal to 1.
  - Otherwise (the reference picture is marked as "reference base picture"), useFlag is set equal to 0.
- Otherwise, if the reference picture is a short-term reference picture, the following applies.
  - If the short-term reference picture is marked as "reference base picture" or if there does not exist a short-term reference picture with an identical value frame_num coded in slice header(s) with dependency_id equal to currDependencyId that is marked as "reference base picture", useFlag is set equal to 1.
  - Otherwise (the short-term picture is not marked as "reference base picture" and there exists a short-term reference picture with an identical value frame_num coded in slice header(s) with dependency_id equal to currDependencyId that is marked as "reference base picture"), useFlag is set equal to 0.
- Otherwise (useRefBasePicFlag is equal to 1 and the reference picture is a long-term reference picture), the following applies.
  - If the long-term reference picture is marked as "reference base picture" or if there does not exist a long-term reference picture with an identical value of LongTermFrameIdx that is marked as "reference base picture", useFlag is set equal to 1.
  - Otherwise (the long-term picture is not marked as "reference base picture" and there exists a long-term reference picture with an identical value of LongTermFrameIdx that is marked as "reference base picture"), useFlag is set equal to 0.

To each short-term reference picture the variables FrameNum and FrameNumWrap are assigned as follows.

- If useFlag is equal to 0, FrameNum and FrameNumWrap for the short-term picture are marked as "unspecified".
- Otherwise, FrameNum is set equal to the syntax element frame_num that has been decoded in the slice header(s) with dependency_id equal to currDependencyId of the corresponding short-term reference picture. And the variable FrameNumWrap is derived by

```
if( FrameNum > frame_num )
        FrameNumWrap = FrameNum – maxFrameNum                    (G-89)
else
        FrameNumWrap = FrameNum
```

where the value of frame_num used in Equation G-89 is the frame_num in the slice header(s) with dependency_id equal to currDependencyId for the current access unit.

NOTE – The values of FrameNum and FrameNumWrap are dependent on the value of currDependencyId. For the calculation of FrameNum and FrameNumWrap only syntax elements of NAL units with dependency_id equal to currDependencyId are used.

Each long-term reference picture has an associated value of LongTermFrameIdx (that was assigned to it as specified in subclause 8.2.5). When useFlag is equal to 0 for a long-term picture, LongTermFrameIdx is temporarily marked as "unspecified". This marking is removed after the invocation of the SVC decoding process for reference picture lists construction as specified in subclause G.8.2.3 or the invocation of the SVC decoded reference picture marking process as specified in subclause G.8.2.4.

To each short-term reference picture a variable PicNum is assigned, and to each long-term reference picture a variable LongTermPicNum is assigned. The values of these variables depend on the value of field_pic_flag and bottom_field_flag for the current slice with dependency_id equal to currDependencyId and they are set as follows.

- If field_pic_flag is equal to 0, the following applies.
  - For each short-term reference frame or complementary reference field pair the following applies.
    - If FrameNumWrap is marked as "unspecified", PicNum is marked as "unspecified".
    - Otherwise (FrameNumWrap is not marked as "unspecified"), PicNum is derived by

      PicNum = FrameNumWrap (G-90)

  - For each long-term reference frame or long-term complementary reference field pair the following applies
    - If LongTermFrameIdx is marked as "unspecified", LongTermPicNum is marked as "unspecified".
    - Otherwise (LongTermFrameIdx is not marked as "unspecified"), LongTermPicNum is derived by

      LongTermPicNum = LongTermFrameIdx (G-91)

- Otherwise (field_pic_flag is equal to 1), the following applies.
  - Let current field refer to the slices of the current access unit with dependency_id equal to currDependencyId.
  - For each short-term reference field the following applies.
    - If FrameNumWrap is marked as "unspecified", PicNum is marked as unspecified.
    - Otherwise (FrameNumWrap is not marked as "unspecified"), the following applies.
      - If the reference field has the same parity as the current field

        PicNum = 2 * FrameNumWrap + 1 (G-92)

      - Otherwise (the reference field has the opposite parity of the current field),

        PicNum = 2 * FrameNumWrap (G-93)

  - For each long-term reference field the following applies.
    - If LongTermFrameIdx is marked as "unspecified", LongTermPicNum is marked as "unspecified".
    - Otherwise (LongTermFrameIdx is not marked as "unspecified"), the following applies.
      - If the reference field has the same parity as the current field

        LongTermPicNum = 2 * LongTermFrameIdx + 1 (G-94)

– Otherwise (the reference field has the opposite parity of the current field),

LongTermPicNum = 2 * LongTermFrameIdx (G-95)

### G.8.2.3 SVC decoding process for reference picture lists construction

This process is invoked at the beginning of decoding each P, EP, B, or EB slice.

Inputs to this process are

– a variable currDependencyId

– a variable useRefLayerRepFlag

– the current slice currSlice

Outputs of this process are

– a reference picture list refPicList0

– for B and EB slices, a reference picture list refPicList1

After applying the process described in this subclause, the output reference picture lists refPicList0 and refPicList1 (when applicable) shall not contain any pictures for which the syntax element temporal_id is greater than the syntax element temporal_id of the current picture.

From here to the end of this subclause, the modifications a) and b) specified in subclause G.8.2 apply.

A variable biPred is derived as follows:

– If the current slice currSlice is a B or EB slice, biPred is set equal to 1.

– Otherwise, biPred is set equal to 0.

Decoded reference pictures are marked as "used for short-term reference" or "used for long-term reference" as specified by the bitstream and specified in subclause G.8.2.4. Short-term reference pictures are identified by the value of frame_num that is decoded in the slice header(s) with dependency_id equal to currDependencyId. Long-term reference pictures are identified by a long-term frame index as specified by the bitstream and specified in subclause G.8.2.4.

Subclause G.8.2.2 is invoked with currDependencyId and useRefBasePicFlag as inputs to specify

– the assignment of variables FrameNum, FrameNumWrap, and PicNum to each of the short-term reference pictures, and

– the assignment of variables LongTermPicNum to each of the long-term reference pictures.

NOTE – During the invocation of subclause G.8.2.2 the variables FrameNum, FrameNumWrap, PicNum, and LongTermPicNum can be marked as "unspecified". Furthermore, the variables LongTermFrameIdx can be temporarily marked as "unspecified" for this subclause and all invocations inside this subclause.

Reference pictures are addressed through reference indices as specified in subclause 8.4.2.1. A reference index is an index into a reference picture list. When biPred is equal to 0, a single reference picture list refPicList0 is constructed. When decoding a B or EB slice (biPred is equal to 1), a second independent reference picture list refPicList1 is constructed in addition to refPicList0.

At the beginning of decoding of each slice, reference picture list refPicList0, and for biPred equal to 1 refPicList1, are derived as follows.

– Initial reference picture lists RefPicList0 and, for biPred equal to 1, RefPicList1 are derived by invoking the initialisation process for reference picture lists as specified in subclause 8.2.4.2. During the initialisation process in subclause 8.2.4.2 all short-term reference pictures for which FrameNum, FrameNumWrap, and PicNum have been marked as "unspecified" by the invocation of subclause G.8.2.2 and all long-term pictures for which LongTermFrameIdx and LongTermPicNum have been marked as "unspecified" by the invocation of subclause G.8.2.2 are considered as non-existing. The reference picture lists are constructed by using only short-term pictures with values of FrameNum, FrameNumWrap, and PicNum that are not marked as "unspecified" and long-term pictures with values of LongTermFrameIdx and LongTermPicNum that are not marked as "unspecified".

– The initial reference picture list RefPicList0 and for biPred equal to 1 RefPicList1 are modified by invoking the reordering process for reference picture lists as specified in subclause 8.2.4.3. During the reordering process in subclause 8.2.4.2 all short-term reference pictures for which FrameNum, FrameNumWrap, and PicNum have been marked as "unspecified" by the invocation of subclause G.8.2.2 and all long-term pictures for which LongTermFrameIdx and LongTermPicNum have been marked as "unspecified" by the invocation of subclause

G.8.2.2 are considered as non-existing. Short-term pictures with values of FrameNum, FrameNumWrap, and PicNum that are marked as "unspecified" and long-term pictures with values of LongTermFrameIdx and LongTermPicNum that are marked as "unspecified" shall not be referred to by reference picture list reordering commands.

– RefPicList0 is assigned to refPicList0.

– When biPred is equal to 1, RefPicList1 is assigned to refPicList1.

NOTE – By the invocation of the process in subclause G.8.2.2 the parameters of some reference picture might have been marked as "unspecified". Since, these pictures are not considered in the construction process for reference picture lists, the reference picture lists refPicList0 and for biPred equal to 1 refPicList1 are dependent on the value of the input parameter currDependencyId.

The number of entries in the modified reference picture list refPicList0 is num_ref_idx_l0_active_minus1 + 1, and for biPred equal to 1 the number of entries in the modified reference picture list refPicList1 is num_ref_idx_l1_active_minus1 + 1. A reference picture may appear at more than one index in the modified reference picture lists refPicList0 or refPicList1.

For all long-term reference pictures for which LongTermFrameIdx has been marked as "unspecified" during the invocation of subclause G.8.2.2, this marking of LongTermFrameIdx is removed.

#### G.8.2.4 SVC decoded reference picture marking process

For all dependency representations of the current access unit, when nal_ref_idc is greater than 0 for the slice data NAL units of the dependency representation, the SVC reference picture marking process for a dependency representation as specified in subclause G.8.2.4.1 is invoked. For these invocations of the process specified in subclause G.8.2.4.1, the modifications a) and b) specified in subclause G.8.2 apply with currDependencyId being equal to dependency_id for the corresponding dependency representation.

#### G.8.2.4.1 SVC reference picture marking process for a dependency representation

Input to this process is a variable currDependencyId.

Output of this process is a modified reference picture marking for dependency representations with dependency_id equal to currDependencyId.

This process is invoked for a decoded picture when nal_ref_idc is not equal to 0 for the dependency representation with dependency_id being equal to currDependencyId.

All syntax elements of the dec_ref_pic_marking( ) syntax structure and the dec_ref_base_pic_marking( ) syntax structure that are referred to by the process specified in this subclause and all child processes invoked from this process are syntax elements of the slice headers with dependency_id equal to currDependencyId and quality_id equal to 0.

NOTE – Coded slice NAL units with quality_id greater than 0 do not contain the syntax structures dec_ref_pic_marking( ) and dec_ref_base_pic_marking( ). For these NAL units, the syntax elements specifying the decoded reference picture marking are inferred from coded slice NAL units with dependency_id equal to DependencyIdMax and quality_id equal to 0. Coded slice NAL units with dependency_id equal to 0 and quality_id equal to 0 do not contain the syntax structure dec_ref_base_pic_marking( ). For these NAL units, the syntax elements of the syntax structure dec_ref_base_pic_marking( ) are inferred from the associated prefix NAL units, which directly precedes the coded slice NAL unit in decoding order.

A decoded picture with nal_ref_idc not equal to 0, referred to as a reference picture, is marked as "used for short-term reference" or "used for long-term reference". When store_ref_base_pic_flag is equal to 1, a second representation of the decoded picture also referred to as reference base picture is marked as "used for short-term reference" or "used for long-term reference" and additionally marked as "reference base picture". Pictures that are marked as "reference base picture" are only used as references for inter prediction of following pictures with use_ref_base_pic_flag equal to 1. These pictures are not used for inter prediction of pictures with use_ref_base_pic_flag equal to 0, and these pictures do not represent an output of the decoding process.

For a decoded reference frame, both of its fields are marked the same as the frame. For a complementary reference field pair, the pair is marked the same as both of its fields. A picture that is marked as "used for short-term reference" is identified by its FrameNum and, when it is a field, by its parity, and, when it is a reference base picture, by the marking "reference base picture". A picture that is marked as "used for long-term reference" is identified by its LongTermFrameIdx and, when it is a field, by its parity, and, when it is a reference base picture, by the marking "reference base picture".

While decoded pictures are represented by the sample arrays $S_L$ and, when ChromaArrayType is not equal to 0, $S_{Cb}$ and $S_{Cr}$, reference base pictures are represented by the sample arrays $B_L$ and, when ChromaArrayType is not equal to 0, $B_{Cb}$ and $B_{Cr}$. When reference base pictures are referenced in the inter prediction process via subclause 8.4.2.1, the samples arrays $B_L$, $B_{Cb}$, and $B_{Cr}$ are referred to as $S_L$, $S_{Cb}$, and $S_{Cr}$, respectively. Reference base pictures are associated with the same descriptive information such as the variables FrameNum, FrameNumWrap, PicNum, LongTermFrameIdx, and

LongTermPicNum as the corresponding decoded picture that is marked by the same invocation of the decoded reference picture marking process.

Frames or complementary field pairs marked as "used for short-term reference" or as "used for long-term reference" can be used as a reference for inter prediction when decoding a frame until the frame, the complementary field pair, or one of its constituent fields is marked as "unused for reference". A field marked as "used for short-term reference" or as "used for long-term reference" can be used as a reference for inter prediction when decoding a field until marked as "unused for reference".

A picture can be marked as "unused for reference" by the sliding window reference picture marking process, a first-in, first-out mechanism specified in subclause 8.2.5.3, or by the adaptive memory control reference picture marking process, a customised adaptive marking operation specified in subclause 8.2.5.4.

A short-term reference picture is identified for use in the decoding process by its variables FrameNum and FrameNumWrap and its picture number PicNum, and, when it is a reference base picture, by the marking as "reference base picture". A long-term reference picture is identified for use in the decoding process by its variable LongTermFrameIdx, its long-term picture number LongTermPicNum, and, when it is a reference base picture, by the marking as "reference base picture". When the current picture is not an IDR picture, the assignment of the variables FrameNum, FrameNumWrap, PicNum, LongTermFrameIdx, and LongTermPicNum for the reference pictures is specified as follows.

– The variables FrameNum, FrameNumWrap, PicNum, LongTermFrameIdx, and LongTermPicNum are assigned to all reference pictures marked as "reference base pictures" by invoking subclause G.8.2.2 with currDependencyId set equal to DependencyIdMax and useRefBasePicFlag set equal to 1 as the inputs.

  NOTE – By this first invoking of subclause G.8.2.2 the variables may be also assigned to reference pictures not marked as "reference base picture". For these pictures, the variables are again assigned by the following invoking of subclause G.8.2.2 and the variables that are assigned in both invocations of subclause G.8.2.2 are identical.

– The variables FrameNum, FrameNumWrap, PicNum, LongTermFrameIdx, and LongTermPicNum are assigned to all reference pictures not marked as "reference base picture" by invoking subclause G.8.2.2 with currDependencyId set equal to DependencyIdMax and useRefBasePicFlag set equal to 0 as the inputs.

Decoded reference picture marking proceeds in the following ordered steps.

– All slices of the current access unit are decoded.

– Depending on the current picture, the following applies.

  – If the current picture is an IDR picture, the following applies.

    – All reference pictures are marked as "unused for reference".

    – Depending on long_term_reference_flag, the following applies.

      – If long_term_reference_flag is equal to 0, the following applies.

        – The IDR picture is marked as "used for short-term reference" and MaxLongTermFrameIdx is set equal to "no long-term frame indices"

        – When store_ref_base_pic_flag is equal to 1, the reference base picture of the IDR picture is marked as "used for short-term reference" and as "reference base picture".

      – Otherwise (long_term_reference_flag is equal to 1), the following applies.

        – The IDR picture is marked as "used for long-term reference", the LongTermFrameIdx for the IDR picture is set equal to 0, and MaxLongTermFrameIdx is set equal to 0.

        – When store_ref_base_pic_flag is equal to 1, the reference base picture of the IDR picture is marked as "used for long-term reference" and as "reference base picture", and the LongTermFrameIdx for the IDR picture is set equal to 0.

  – Otherwise (the current picture is not an IDR picture), the following applies.

    – When store_ref_base_pic_flag is equal to 1, the following applies.

      – If adaptive_ref_base_pic_marking_mode_flag is equal to 1, the adaptive memory control decoded reference picture marking process as specified in subclause 8.2.5.4 is invoked with the syntax elements from the syntax structure dec_ref_base_pic_marking( ) and adaptive_ref_base_pic_marking_mode_flag replacing adaptive_ref_pic_marking_mode_flag. Only the reference pictures that are marked as "reference base picture" are considered in this invocation of the process.

- Otherwise (adaptive_ref_base_pic_marking_mode_flag is equal to 0), the sliding window decoded reference picture marking process as specified in subclause 8.2.5.3 is invoked with adaptive_ref_base_pic_marking_mode_flag replacing adaptive_ref_pic_marking_mode_flag. When during the invocation of this process there is a decoded picture not marked as "reference base picture" and a reference base picture marked as "reference base picture", and both of which have the smallest value of FrameNumWrap, then only the picture marked as "reference base picture" is marked as "unused for reference".
- The reference base picture of the current decoded picture is be marked as "used for short-term reference" and as "reference base picture".

– Depending on adaptive_ref_pic_marking_mode_flag, the following applies.
  - If adaptive_ref_pic_marking_mode_flag is equal to 0, the following applies.
    - The sliding window decoded reference picture marking process as specified in subclause 8.2.5.3 is invoked. When during the invocation of this process there is a decoded picture not marked as "reference base picture" and a reference base picture marked as "reference base picture", and both of which have the smallest value of FrameNumWrap, then only the reference base picture is marked as "unused for reference".
    - The current decoded picture is marked as "used for short-term reference".
  - Otherwise (adaptive_ref_pic_marking_mode_flag is equal to 1), the following applies.
    - The adaptive memory control decoded reference picture marking process as specified in subclause 8.2.5.4 with the syntax elements from the dec_ref_pic_marking( ) syntax structure is invoked. Only the reference pictures that are not marked as "reference base picture" are considered in the invocation of the process. Additionally, the following applies.
      - All pictures for which LongTermFrameIdx is greater than MaxLongTermFrameIdx and that are marked as "used for long-term reference" and "reference base picture" are marked as "unused for reference".
      - When during the invocation of this process a LongTermFrameIdx is assigned to a picture and there exists a "reference base picture" with the same value of PicNum (before assigning the long-term frame index), the LongTermFrameIdx is also assigned to the reference base picture, and the marking is changed from "used for short-term reference" to "used for long-term reference".
    - When the current picture was not marked as "used for long-term reference" by memory_management_control_operation equal to 6, it is marked as "used for short-term reference".

NOTE – When store_ref_base_pic_flag is equal to 1, num_ref_frames shall be greater than or equal to 2.

When the adaptive memory control decoded reference picture marking process as specified in subclause 8.2.5.4 is invoked to mark reference pictures as "unused for reference" after decoding of a reference picture with temporal_id equal to tL, no reference picture with temporal_id less than tL shall be marked as "unused for reference".

For all long-term reference pictures for which LongTermFrameIdx has been marked as "unspecified" during the invocation of subclause G.8.2.2, this marking of LongTermFrameIdx is removed.

#### G.8.2.5 SVC decoding process for gaps in frame_num

For all dependency representations of the current access unit, the following applies.

– The syntax elements gaps_in_frame_num_value_allowed_flag and frame_num and the derived variables PrevRefFrameNum and MaxFrameNum are the syntax elements and derived upper-case variables for the considered dependency representation.

– When gaps_in_frame_num_value_allowed_flag is equal to 0, the bitstream shall not contain data that result in frame_num not being equal to PrevRefFrameNum or ( PrevRefFrameNum + 1 ) % MaxFrameNum.

  NOTE – When gaps_in_frame_num_value_allowed_flag is equal to 0 and frame_num is not equal to PrevRefFrameNum and is not equal to ( PrevRefFrameNum + 1 ) % MaxFrameNum, the decoding process should infer an unintentional loss of pictures.

– When frame_num is not equal to PrevRefFrameNum and is not equal to ( PrevRefFrameNum + 1 ) % MaxFrameNum, the decoding process for gaps in frame_num as specified in subclause 8.2.5.2 is invoked. For this invocation of the process specified in subclause 8.2.5.2, the modifications a) and b) specified in subclause G.8.2 apply with currDependencyId being equal to dependency_id for the considered dependency representation.

### G.8.3 SVC intra decoding processes

Subclause G.8.3.1 specifies the SVC derivation process for intra prediction modes.

Subclause G.8.3.2 specifies the SVC intra sample prediction and construction process.

#### G.8.3.1 SVC derivation process for intra prediction modes

This process is only invoked when base_mode_flag is equal to 0 and mbType[ CurrMbAddr ] specified as input to this process is equal to I_PCM, I_16x16, I_8x8, or I_4x4.

Inputs to this process are

- a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,
- a list fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,
- a list mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,
- a (PicSizeInMbs)x16 array ipred4x4 specifying Intra_4x4 prediction modes for macroblocks of the current layer representation,
- a (PicSizeInMbs)x4 array ipred8x8 specifying Intra_8x8 prediction modes for macroblocks of the current layer representation,
- a list intra16x16 with PicSizeInMbs elements specifying Intra_16x16 prediction modes for macroblocks of the current layer representation,
- when ChromaArrayType is equal to 1 or 2, a list intraChroma with PicSizeInMbs elements specifying intra chroma prediction modes for macroblocks of the current layer representation.

Outputs of this process are

- a modified version of the array ipred4x4,
- a modified version of the array ipred8x8,
- a modified version of the array ipred16x16,
- when ChromaArrayType is equal to 1 or 2, a modified version of the array ipredChroma.

For all processes specified in clause 6 that are invoked from the process specified in this subclause or a child process of the process specified in this subclause, the following modifications apply.

a) In subclause 6.4.11.2, a macroblock with address mbAddr is treated as field macroblock when fieldMbFlag[ mbAddr ] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[ mbAddr ] is equal to 0. In particular, the current macroblock is treated as field macroblock when fieldMbFlag[ CurrMbAddr ] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[ CurrMbAddr ] is equal to 0.

b) In subclause 6.4.7, a macroblock with address mbAddr is treated to belong to a different slice than the current macroblock CurrMbAddr, when sliceIdc[ mbAddr ] is not equal to sliceIdc[ CurrMbAddr ].

c) In subclause 6.4.11.2, a macroblock mbAddr is treated as top macroblock when (mbAddr % 2) is equal to 0, and it is treated as bottom macroblock when (mbAddr % 2) is equal to 1.

When mbType[ CurrMbAddr ] is not equal to I_PCM, the following applies.

- If mbType[ CurrMbAddr ] is equal to I_4x4, the SVC derivation process for Intra_4x4 prediction modes as specified in subclause G.8.3.1.1 is invoked with sliceIdc, fieldMbFlag, mbType, ipred4x4, and ipred8x8 as the inputs and the output is a modified version of the array ipred4x4.
- Otherwise, if mbType[ CurrMbAddr ] is equal to I_8x8, the SVC derivation process for Intra_8x8 prediction modes as specified in subclause G.8.3.1.2 is invoked with sliceIdc, fieldMbFlag, mbType, ipred4x4, and ipred8x8 as the inputs and the output is a modified version of the array ipred8x8.
- Otherwise, if mbType[ CurrMbAddr ] is equal to I_16x16, ipred16x16[ CurrMbAddr ] is set equal to Intra16x16PredMode.

When ChromaArrayType is equal to 1 or 2 and mbType[ CurrMbAddr ] is not equal to I_PCM, ipredChroma[ CurrMbAddr ] is set equal to intra_chroma_pred_mode.

#### G.8.3.1.1 SVC derivation process for Intra_4x4 prediction modes

Inputs to this process are

- a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,
- a list fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,
- a list mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,
- a (PicSizeInMbs)x16 array ipred4x4 specifying Intra_4x4 prediction modes for macroblocks of the current layer representation,
- a (PicSizeInMbs)x4 array ipred8x8 specifying Intra_8x8 prediction modes for macroblocks of the current layer representation.

Output of this process is a modified version of the array ipred4x4.

The 4x4 blocks indexed by c4x4BlkIdx = 0..15 are processed in increasing order of c4x4BlkIdx, and for each 4x4 block, the following ordered steps are specified.

1. The derivation process for neighbouring 4x4 luma blocks as specified in subclause 6.4.10.4 is invoked with c4x4BlkIdx as the input and the outputs are assigned to mbAddrA, c4x4BlkIdxA, mbAddrB, and c4x4BlkIdxB. For this invocation of the process in subclause 6.4.10.4, the modifications specified in items a) through c) in subclause G.8.3.1 apply.
2. For N being replaced by A and B, the variables availableFlagN are derived as follows.
   - If the macroblock mbAddrN is available and mbType[ mbAddrN ] is equal to I_PCM, I_16x16, I_8x8, or I_4x4, availableFlagN is set equal to 1.
   - Otherwise (macroblock mbAddrN is not available or mbType[ mbAddrN ] is not equal to I_PCM, I_16x16, I_8x8, or I_4x4), availableFlagN is set equal to 0.
3. The variable dcPredModePredictedFlag is derived as follows.
   - If availableFlagA or availableFlagB is equal to 0, dcPredModePredictedFlag is set equal to 1.
   - Otherwise (availableFlagA is equal to 1 and availableFlagB is equal to 1), dcPredModePredictedFlag is set equal to 0.
4. For N being replaced by A and B, the variables intraMxMPredModeN are derived as follows.
   - If dcPredModePredictedFlag is equal to 0 and mbType[ mbAddrN ] is equal to I_4x4, intraMxMPredModeN is set equal to ipred4x4[ mbAddrN ][ c4x4BlkIdxN ].
   - Otherwise, if dcPredModePredictedFlag is equal to 0 and mbType[ mbAddrN ] is equal to I_8x8, intraMxMPredModeN is set equal to ipred8x8[ mbAddrN ][ c4x4BlkIdxN >> 2 ].
   - Otherwise (dcPredModePredictedFlag is equal to 1 or (mbType[ mbAddrN ] is not equal to I_4x4 and mbType[ mbAddrN ] is not equal to I_8x8)), intraMxMPredModeN is set equal to 2.
5. The element ipred4x4[ CurrMbAddr ][ c4x4BlkIdx ] of the array ipred4x4 is derived by applying the procedure specified in the following pseudo-code.

```
predIntra4x4PredMode = Min( intraMxMPredModeA, intraMxMPredModeB )
if( prev_intra4x4_pred_mode_flag[ c4x4BlkIdx ] )
   ipred4x4[ CurrMbAddr ][ c4x4BlkIdx ] = predIntra4x4PredMode
else if( rem_intra4x4_pred_mode[ c4x4BlkIdx ] < predIntra4x4PredMode )          (G-96)
   ipred4x4[ CurrMbAddr ][ c4x4BlkIdx ] = rem_intra4x4_pred_mode[ c4x4BlkIdx ]
else
   ipred4x4[ CurrMbAddr ][ c4x4BlkIdx ] = rem_intra4x4_pred_mode[ c4x4BlkIdx ] + 1
```

#### G.8.3.1.2 SVC derivation process for Intra_8x8 prediction modes

Inputs to this process are

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,

– a list fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a list mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a (PicSizeInMbs)x16 array ipred4x4 specifying Intra_4x4 prediction modes for macroblocks of the current layer representation,

– a (PicSizeInMbs)x4 array ipred8x8 specifying Intra_8x8 prediction modes for macroblocks of the current layer representation.

Output of this process is a modified version of the array ipred8x8.

The 8x8 blocks indexed by c8x8BlkIdx = 0..3 are processed in increasing order of c8x8BlkIdx, and for each 8x8 block, the following ordered steps are specified.

1. The derivation process for neighbouring 8x8 luma blocks as specified in subclause 6.4.10.2 is invoked with c8x8BlkIdx as the input and the outputs are assigned to mbAddrA, c8x8BlkIdxA, mbAddrB, and c8x8BlkIdxB. For this invocation of the process in subclause 6.4.10.2, the modifications specified in items a) through c) in subclause G.8.3.1 apply.

2. For N being replaced by A and B, the variables availableFlagN are derived as follows.

   – If the macroblock mbAddrN is available and mbType[ mbAddrN ] is equal to I_PCM, I_16x16, I_8x8, or I_4x4, availableFlagN is set equal to 1.

   – Otherwise (macroblock mbAddrN is not available or mbType[ mbAddrN ] is not equal to I_PCM, I_16x16, I_8x8, or I_4x4), availableFlagN is set equal to 0.

3. The variable dcPredModePredictedFlag is derived as follows.

   – If availableFlagA or availableFlagB is equal to 0, dcPredModePredictedFlag is set equal to 1.

   – Otherwise (availableFlagA is equal to 1 and availableFlagB are equal to 1), dcPredModePredictedFlag is set equal to 0.

4. For N being replaced by A and B, the variables intraMxMPredModeN are derived as follows.

   – If dcPredModePredictedFlag is equal to 0 and mbType[ mbAddrN ] is equal to I_4x4, intraMxMPredModeN is set equal to ipred4x4[ mbAddrN ][ c8x8BlkIdxN * 4 + c4x4Idx ] with the variable c4x4Idx being derived as follows.

     – If N is equal to B, c4x4Idx is set equal to 2.

     – Otherwise, if fieldMbFlag[ CurrMbAddr ] is equal to 0, fieldMbFlag[ mbAddrN ] is equal to 1, and c8x8BlkIdx is equal to 2, c4x4Idx is set equal to 3.

     – Otherwise (N is equal to A and (fieldMbFlag[ CurrMbAddr ] is equal to 1 or fieldMbFlag[ mbAddrN ] is equal to 0 or c8x8BlkIdx is not equal to 2)), c4x4Idx is set equal to 1.

   – Otherwise, if dcPredModePredictedFlag is equal to 0 and mbType[ mbAddrN ] is equal to I_8x8, intraMxMPredModeN is set equal to ipred8x8[ mbAddrN ][ c8x8BlkIdxN ].

   – Otherwise (dcPredModePredictedFlag is equal to 1 or (mbType[ mbAddrN ] is not equal to I_4x4 and mbType[ mbAddrN ] is not equal to I_8x8)), intraMxMPredModeN is set equal to 2.

5. The element ipred8x8[ CurrMbAddr ][ c8x8BlkIdx ] of the array ipred8x8 is derived by applying the procedure specified in the following pseudo-code.

```
predIntra8x8PredMode = Min( intraMxMPredModeA, intraMxMPredModeB )
if( prev_intra8x8_pred_mode_flag[ c8x8BlkIdx ] )
   ipred8x8[ CurrMbAddr ][ c8x8BlkIdx ] = predIntra8x8PredMode
else if( rem_intra8x8_pred_mode[ c8x8BlkIdx ] < predIntra8x8PredMode )          (G-97)
   ipred8x8[ CurrMbAddr ][ c8x8BlkIdx ] = rem_intra8x8_pred_mode[ c8x8BlkIdx ]
else
   ipred8x8[ CurrMbAddr ][ c8x8BlkIdx ] = rem_intra8x8_pred_mode[ c8x8BlkIdx ] + 1
```

### G.8.3.2 SVC intra sample prediction and construction process

This process is only invoked when mbType specified as input to this process is equal to I_PCM, I_16x16, I_8x8, or I_4x4.

Inputs to this process are

- a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,
- a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,
- a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,
- a list ipred4x4 with 16 elements specifying Intra_4x4 prediction modes for the current macroblock,
- a list ipred8x8 with 4 elements specifying Intra_8x8 prediction modes for the current macroblock,
- a variable ipred16x16 specifying the Intra_16x16 prediction mode for the current macroblock,
- a variable ipredChroma specifying the intra chroma prediction mode for the current macroblock,
- a variable cTrafo specifying the transform type for the current macroblock,
- a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,
- a (PicWidthInSamples$_L$)x(PicHeightInSamples$_L$) array picSamples$_L$ containing constructed luma sample values for the current layer representation,
- when ChromaArrayType is not equal to 0, two (PicWidthInSamples$_C$)x(PicHeightInSamples$_C$) arrays picSamples$_{Cb}$ and picSamples$_{Cr}$ containing constructed chroma sample values for the current layer representation.

Outputs of this process are

- a modified version of the array picSamples$_L$,
- when ChromaArrayType is not equal to 0, modified versions of the arrays picSamples$_{Cb}$ and picSamples$_{Cr}$.

For all processes specified in clauses 6 or 8 that are invoked from the process specified in this subclause or a child process of the process specified in this subclause, the following modifications apply.

a) In subclause 6.4.11.2, a macroblock with address mbAddr is treated as field macroblock when fieldMbFlag[ mbAddr ] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[ mbAddr ] is equal to 0. In particular, the current macroblock is treated as field macroblock when fieldMbFlag[ CurrMbAddr ] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[ CurrMbAddr ] is equal to 0.

b) In subclause 6.4.7, a macroblock with address mbAddr is treated to belong to a different slice than the current macroblock CurrMbAddr, when sliceIdc[ mbAddr ] is not equal to sliceIdc[ CurrMbAddr ].

c) In subclause 6.4.11.2, a macroblock mbAddr is treated as top macroblock when (mbAddr % 2) is equal to 0, and it is treated as bottom macroblock when (mbAddr % 2) is equal to 1.

d) In subclauses 8.3.1.2, 8.3.2.2, 8.3.3, and 8.3.4, the variables Intra4x4PredMode, Intra8x8PredMode, Intra16x16PredMode, and intra_chroma_pred_mode are replaced by ipred4x4, ipred8x8, ipred16x16, and ipredChroma, respectively.

e) In subclauses 8.3.1.2, 8.3.2.2, 8.3.3, and 8.3.4, the syntax element mb_type of a macroblock with macroblock address mbAddr is replaced by mbType[ mbAddr ].

f) In subclauses 8.3.1.2, 8.3.2.2, 8.3.3, and 8.3.4, a macroblock with mbAddrN is treated as coded in Inter prediction mode when mbType[ mbAddrN ] is not equal to I_PCM, I_16x16, I_8x8, or I_4x4.

The SVC intra sample prediction and construction process proceeds in the following ordered steps.

1. The construction process for luma residuals or chroma residuals with ChromaArrayType equal to 3 as specified in subclause G.8.5.3.1 is invoked with cTrafo and sTCoeff as the inputs and the outputs are residual luma sample values as a 16x16 array mbRes$_L$ with elements mbRes$_L$[ x, y ].
2. When ChromaArrayType is not equal to 0, the construction process for chroma residuals as specified in subclause G.8.5.3.2 is invoked with cTrafo and sTCoeff as the inputs and the outputs are residual chroma

sample values as two (MbWidthC)x(MbHeightC) arrays $mbRes_{Cb}$ and $mbRes_{Cr}$ with elements $mbRes_{Cb}[ x, y ]$ and $mbRes_{Cr}[ x, y ]$, respectively.

3. The SVC intra prediction and construction process for luma samples or chroma samples with ChromaArrayType equal to 3 as specified in subclause G.8.3.2.1 is invoked with $BitDepth_Y$, sliceIdc, fieldMbFlag, mbType, ipred4x4, ipred8x8, ipred16x16, $mbRes_L$, and $picSamples_L$ as the inputs and the output is a modified version of the array $picSamples_L$.

4. When ChromaArrayType is not equal to 0, the SVC intra prediction and construction process for chroma samples as specified in subclause G.8.3.2.2 is invoked with sliceIdc, fieldMbFlag, mbType, ipred4x4, ipred8x8, ipred16x16, ipredChroma, $mbRes_{Cb}$, $mbRes_{Cr}$, $picSamples_{Cb}$, and $picSamples_{Cr}$ as the inputs and the outputs are modified versions of the arrays $picSamples_{Cb}$ and $picSamples_{Cr}$.

#### G.8.3.2.1 SVC intra prediction and construction process for luma samples or chroma samples with ChromaArrayType equal to 3

Inputs to this process are

- a variable bitDepth specifying the bit depth,
- a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,
- a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,
- a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,
- a list ipred4x4 with 16 elements specifying Intra_4x4 prediction modes for the current macroblock,
- a list ipred8x8 with 4 elements specifying Intra_8x8 prediction modes for the current macroblock,
- a variable ipred16x16 specifying the Intra_16x16 prediction mode for the current macroblock,
- a 16x16 array mbRes containing residual sample values for the current macroblock,
- a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array picSamples containing constructed sample values for the current layer representation.

Outputs of this process is a modified version of the array picSamples.

Depending on mbType[ CurrMbAddr ], the following applies.

- If mbType[ CurrMbAddr ] is equal to I_PCM, the SVC construction process for luma samples and chroma samples with ChromaArrayType equal to 3 of I_PCM macroblocks as specified in subclause G.8.3.2.1.1 is invoked with fieldMbFlag, mbRes, and picSamples as the inputs and the output is a modified version of the array picSamples.
- Otherwise, if mbType[ CurrMbAddr ] is equal to I_4x4, the SVC Intra_4x4 sample prediction and construction process as specified in subclause G.8.3.2.1.2 is invoked with bitDepth, sliceIdc, fieldMbFlag, mbType, ipred4x4, mbRes, and picSamples as the inputs and the output is a modified version of the array picSamples.
- Otherwise, if mbType[ CurrMbAddr ] is equal to I_8x8, the SVC Intra_8x8 sample prediction and construction process as specified in subclause G.8.3.2.1.3 is invoked with bitDepth, sliceIdc, fieldMbFlag, mbType, ipred8x8, mbRes, and picSamples as the inputs and the output is a modified version of the array picSamples.
- Otherwise (mbType[ CurrMbAddr ] is equal to I_16x16), the SVC Intra_16x16 sample prediction and construction process as specified in subclause G.8.3.2.1.4 is invoked with bitDepth, sliceIdc, fieldMbFlag, mbType, ipred16x16, mbRes, and picSamples as the inputs and the output is a modified version of the array picSamples.

##### G.8.3.2.1.1 SVC construction process for luma samples and chroma samples with ChromaArrayType equal to 3 of I_PCM macroblocks

Inputs to this process are

- a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,
- a 16x16 array mbRes containing residual sample values for the current macroblock,
- a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array picSamples containing constructed sample values for the current layer representation.

Outputs of this process is a modified version of the array picSamples.

The picture sample array construction process for a signal component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag[ CurrMbAddr ], mbW set equal to 16, mbH set equal to 16, mbRes, and picSamples as the inputs and the output is a modified version of the array picSamples.

#### G.8.3.2.1.2 SVC Intra_4x4 sample prediction and construction process

Inputs to this process are

- a variable bitDepth specifying the bit depth,
- a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,
- a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,
- a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,
- a list ipred4x4 with 16 elements specifying Intra_4x4 prediction modes for the current macroblock,
- a 16x16 array mbRes containing residual sample values for the current macroblock,
- a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array picSamples containing constructed sample values for the current layer representation.

Outputs of this process is a modified version of the array picSamples.

Let mbSamples be a 16x16 array containing constructed intra sample values for the current macroblock. All elements of mbSamples are initially set equal to 0.

The 4x4 blocks indexed by c4x4BlkIdx = 0..15 are processed in increasing order of c4x4BlkIdx, and for each 4x4 block, the following ordered steps are specified.

1. The Intra_4x4 sample prediction process as specified in subclause 8.3.1.2 is invoked with c4x4BlkIdx and picSamples as the inputs and the outputs are intra prediction sample values as a 4x4 array pred4x4 with elements pred4x4[ x, y ]. For this invocation of the process in subclause 8.3.1.2, the modifications specified in items a) through f) of subclause G.8.3.2 apply. Additionally in subclause 8.3.1.2.3, which may be invoked as part of the process specified in subclause 8.3.1.2, the variable $BitDepth_Y$ is replaced by bitDepth.
2. The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with c4x4BlkIdx as the input and the output is assigned to ( xP, yP ).
3. For x = xP..(xP + 3) and y = yP..(yP + 3) and with Clip( a ) specifying Clip3( 0, ( 1 << bitDepth ) – 1, a ), the elements mbSamples[ x, y ] of the 16x16 array mbSamples are derived by

$$mbSamples[ x, y ] = Clip( pred4x4[ x – xP, y – yP ] + mbRes[ x, y ] ) \quad \text{(G-98)}$$

4. The picture sample array construction process for a signal component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag[ CurrMbAddr ], mbW set equal to 16, mbH set equal to 16, mbSamples, and picSamples as the inputs and the output is a modified version of the array picSamples.

   NOTE – When c4x4BlkIdx is less than 15, the array mbSamples does only contain constructed intra samples for 4x4 blocks with c4x4BlkIdx less than or equal to the current value of c4x4BlkIdx.

#### G.8.3.2.1.3 SVC Intra_8x8 sample prediction and construction process

Inputs to this process are

- a variable bitDepth specifying the bit depth,
- a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,
- a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,
- a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,
- a list ipred8x8 with 4 elements specifying Intra_8x8 prediction modes for the current macroblock,

– a 16x16 array mbRes containing residual sample values for the current macroblock,

– a (PicWidthInSamples$_L$)x(PicHeightInSamples$_L$) array picSamples containing constructed sample values for the current layer representation.

Outputs of this process is a modified version of the array picSamples.

Let mbSamples be a 16x16 array containing constructed intra sample values for the current macroblock. All elements of mbSamples are initially set equal to 0.

The 8x8 blocks indexed by c8x8BlkIdx = 0..3 are processed in increasing order of c8x8BlkIdx, and for each 8x8 block, the following ordered steps are specified.

1. The Intra_8x8 sample prediction process as specified in subclause 8.3.2.2 is invoked with c8x8BlkIdx and picSamples as the inputs and the outputs are intra prediction sample values as an 8x8 array pred8x8 with elements pred8x8[ x, y ]. For this invocation of the process in subclause 8.3.2.2, the modifications specified in items a) through f) of subclause G.8.3.2 apply. Additionally in subclause 8.3.2.2.4, which may be invoked as part of the process specified in subclause 8.3.2.2, the variable BitDepth$_Y$ is replaced by bitDepth.

2. The inverse 8x8 luma block scanning process as specified in subclause 6.4.5 is invoked with c8x8BlkIdx as the input and the output is assigned to ( xP, yP ).

3. For x = xP..(xP + 7) and y = yP..(yP + 7) and with Clip( a ) specifying Clip3( 0, ( 1 << bitDepth ) – 1, a ), the elements mbSamples[ x, y ] of the 16x16 array mbSamples are derived by

$$mbSamples[ x, y ] = Clip( pred8x8[ x – xP, y – yP ] + mbRes[ x, y ] ) \quad \text{(G-99)}$$

4. The picture sample array construction process for a signal component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag[ CurrMbAddr ], mbW set equal to 16, mbH set equal to 16, mbSamples, and picSamples as the inputs and the output is a modified version of the array picSamples.

NOTE – When c8x8BlkIdx is less than 3, the array mbSamples does only contain constructed intra samples for 8x8 blocks with c8x8BlkIdx less than or equal to the current value of c8x8BlkIdx.

#### G.8.3.2.1.4 SVC Intra_16x16 sample prediction and construction process

Inputs to this process are

– a variable bitDepth specifying the bit depth,

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,

– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a variable ipred16x16 specifying the Intra_16x16 prediction mode for the current macroblock,

– a 16x16 array mbRes containing residual sample values for the current macroblock,

– a (PicWidthInSamples$_L$)x(PicHeightInSamples$_L$) array picSamples containing constructed sample values for the current layer representation.

Outputs of this process is a modified version of the array picSamples.

The SVC Intra_16x16 sample prediction and construction process proceeds in the following ordered steps.

1. The Intra_16x16 prediction process for luma samples as specified in subclause 8.3.3 is invoked with picSamples as the input and the outputs are intra prediction sample values as a 16x16 array pred16x16 with elements pred16x16[ x, y ]. For this invocation of the process in subclause 8.3.3, the modifications specified in items a) through f) of subclause G.8.3.2 apply. Additionally in subclause 8.3.3.3, which may be invoked as part of the process specified in subclause 8.3.3, the variable BitDepth$_Y$ is replaced by bitDepth.

2. With Clip( a ) specifying Clip3( 0, ( 1 << bitDepth ) – 1, a ), the 16x16 array mbSamples is derived by

$$mbSamples[ x, y ] = Clip( pred16x16[ x, y ] + mbRes[ x, y ] ) \quad \text{with } x, y = 0..15 \quad \text{(G-100)}$$

3. The picture sample array construction process for a signal component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag[ CurrMbAddr ], mbW set equal to 16, mbH set equal to 16, mbSamples, and picSamples as the inputs and the output is a modified version of the array picSamples.

#### G.8.3.2.2 SVC intra prediction and construction process for chroma samples

Inputs to this process are

- a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,
- a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,
- a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,
- a list ipred4x4 with 16 elements specifying Intra_4x4 prediction modes for the current macroblock,
- a list ipred8x8 with 4 elements specifying Intra_8x8 prediction modes for the current macroblock,
- a variable ipred16x16 specifying the Intra_16x16 prediction mode for the current macroblock,
- a variable ipredChroma specifying the intra chroma prediction mode for the current macroblock,
- two (MbWidthC)x(MbHeightC) arrays $mbRes_{Cb}$ and $mbRes_{Cr}$ containing residual chroma sample values for the current macroblock,
- two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ containing constructed sample values for the current layer representation.

Outputs of this process are modified versions of the arrays $picSamples_{Cb}$ and $picSamples_{Cr}$.

Depending on ChromaArrayType, the following applies.

- If ChromaArrayType is equal to 1 or 2, the following applies.
  - If mbType[ CurrMbAddr ] is equal to I_PCM, the SVC construction process for chroma samples of I_PCM macroblock as specified in subclause G.8.3.2.2.1 is invoked with fieldMbFlag, sTCoeff, $picSamples_{Cb}$, and $picSamples_{Cr}$ as the inputs and the outputs are modified versions of $picSamples_{Cb}$ and $picSamples_{Cr}$.
  - Otherwise (mbType[ CurrMbAddr ] is not equal to I_PCM), the SVC intra prediction and construction process for chroma samples with ChromaArrayType equal to 1 or 2 as specified in subclause G.8.3.2.2.2 is invoked with sliceIdc, fieldMbFlag, mbType, ipredChroma, $mbRes_{Cb}$, $mbRes_{Cr}$, $picSamples_{Cb}$, and $picSamples_{Cr}$ as the inputs and the outputs are modified versions of the arrays $picSamples_{Cb}$ and $picSamples_{Cr}$.
- Otherwise (ChromaArrayType is equal to 3), for CX being replaced by Cb and Cr, the SVC intra prediction and construction process for luma samples or chroma samples with ChromaArrayType equal to 3 as specified in subclause G.8.3.2.1 is invoked with $BitDepth_C$, sliceIdc, fieldMbFlag, mbType, ipred4x4, ipred8x8, ipred16x16, $mbRes_{CX}$, and $picSamples_{CX}$ as the inputs and the output is a modified version of the array $picSamples_{CX}$.

##### G.8.3.2.2.1 SVC construction process for chroma samples of I_PCM macroblocks

Inputs to this process are

- a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,
- two (MbWidthC)x(MbHeightC) arrays $mbRes_{Cb}$ and $mbRes_{Cr}$ containing residual chroma sample values for the current macroblock,
- two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ containing constructed chroma sample values for the current layer representation.

Outputs of this process are modified versions of the arrays $picSamples_{Cb}$ and $picSamples_{Cr}$.

For CX being replaced by Cb and Cr, the picture sample array construction process for a signal component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag[ CurrMbAddr ], mbW set equal to MbWidthC, mbH set equal to MbHeightC, $mbRes_{CX}$, and $picSamples_{CX}$ as the inputs and the output is a modified version of the array $picSamples_{CX}$.

##### G.8.3.2.2.2 SVC intra prediction and construction process for chroma samples with ChromaArrayType equal to 1 or 2

This process is only invoked when ChromaArrayType is equal to 1 or 2.

Inputs to this process are

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,

– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a variable ipredChroma specifying the intra chroma prediction mode for the current macroblock,

– two (MbWidthC)x(MbHeightC) arrays $mbRes_{Cb}$ and $mbRes_{Cr}$ containing residual chroma sample values for the current macroblock,

– two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ containing constructed chroma sample values for the current layer representation.

Outputs of this process are modified versions of the arrays $picSamples_{Cb}$ and $picSamples_{Cr}$.

The SVC intra prediction and construction process for chroma samples with ChromaArrayType equal to 1 or 2 proceeds in the following ordered steps.

1. The intra prediction process for chroma samples as specified in subclause 8.3.4 is invoked with $picSamples_{Cb}$ and $picSamples_{Cr}$ as the inputs and the outputs are intra prediction chroma sample values as two (MbWidthC)x(MbHeightC) arrays $pred_{Cb}$ and $pred_{Cr}$ with elements $pred_{Cb}[ x, y ]$ and $pred_{Cr}[ x, y ]$, respectively. For this invocation of the process in subclause 8.3.4, the modifications specified in items a) through f) of subclause G.8.3.2 apply.

2. For CX being replaced by Cb and Cr, the (MbWidthC)x(MbHeightC) array $mbSamples_{CX}$ is derived by

$$mbSamples_{CX}[ x, y ] = Clip1_C( pred_{CX}[ x, y ] + mbRes_{CX}[ x, y ] ) \quad \text{with } x = 0..(MbWidthC - 1) \text{ and } y = 0..(MbHeightC - 1) \quad \text{(G-101)}$$

3. For CX being replaced by Cb and Cr, the picture sample array construction process for a signal component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag[ CurrMbAddr ], mbW set equal to MbWidthC, mbH set equal to MbHeightC, $mbSamples_{CX}$, and $picSamples_{CX}$ as the inputs and the output is a modified version of the array $picSamples_{CX}$.

### G.8.4 SVC Inter prediction process

Subclause G.8.4.1 specifies the SVC derivation process for motion vector components and reference indices.

Subclause G.8.4.2 specifies the SVC decoding process for Inter prediction samples

#### G.8.4.1 SVC derivation process for motion vector components and reference indices

Inputs to this process are

– a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,

– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a (PicSizeInMbs)x4 array subMbType specifying sub-macroblock types for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4 arrays predFlagL0 and predFlagL1 specifying prediction utilization flags for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4 arrays refIdxL0 and refIdxL1 specifying reference indices for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4x4x2 arrays mvL0 and mvL1 specifying motion vector components for the macroblocks of the current layer representation,

– a one-dimensional array mvCnt with PicSizeInMbs elements specifying the number of motion vectors for the macroblocks of the current layer representation,

– two 2x2 arrays refIdxILPredL0 and refIdxILPredL1 specifying inter-layer reference index predictors for the current macroblock,

– two 4x4x2 arrays mvILPredL0 and mvILPredL1 specifying inter-layer motion vector predictors for the current macroblock,

– when DQId is equal to 0 and (slice_type % 5) is equal to 1, the reference list refPicList1.

Outputs of this process are

– modified versions of the arrays predFlagL0 and predFlagL1,

– modified versions of the arrays refIdxL0 and refIdxL1,

– modified versions of the arrays mvL0 and mvL1,

– a modified version of the array mvCnt.

Depending on mbType[ CurrMbAddr ], the following applies.

– If mbType[ CurrMbAddr ] is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, the arrays predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and mvCnt are modified by

predFlagLX[ CurrMbAddr ][ m ] = 0 with X = 0..1, m = 0..3 (G-102)
refIdxLX[ CurrMbAddr ][ m ] = –1 with X = 0..1, m = 0..3 (G-103)
mvLX[ CurrMbAddr ][ m ][ s ][ c ] = 0 with X = 0..1, m = 0..3, s = 0..3, c = 0..1 (G-104)
mvCnt[ CurrMbAddr ] = 0 (G-105)

– Otherwise (mbType[ CurrMbAddr ] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL), the arrays predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and mvCnt are modified as specified by the following text.

The variable isDirectFlag is derived as follows.

– If any of the following conditions is true, isDirectFlag is set equal to 1.

  – mbType[ CurrMbAddr ] is equal to B_Skip or B_Direct_16x16

  – mbType[ CurrMbAddr ] is equal to B_8x8 and subMbType[ CurrMbAddr ][ mbPartIdx ] is equal to B_Direct_8x8

– Otherwise, isDirectFlag is set equal to 0.

The variable numMbPart is derived as follows.

– If mbType[ CurrMbAddr ] is equal to B_Skip or B_Direct_16x16 and DQId is equal to 0 (nal_unit_type is not equal to 20), numMbPart is set equal to 4.

– Otherwise, if mbType[ CurrMbAddr ] is equal to B_Skip or B_Direct_16x16 (and DQId is greater than 0 and nal_unit_type is equal to 20), numMbPart is set equal to 1.

– Otherwise (mbType[ CurrMbAddr ] is not equal to B_Skip or B_Direct_16x16), numMbPart is set equal to NumMbPart( mbType[ CurrMbAddr ] ).

The macroblock partition index mbPartIdx proceeds over the values 0..(numMbPart – 1), and for each value of mbPartIdx the following ordered steps are specified.

1. The variable numSubMbPart is derived as follows.

   – If isDirectFlag is equal to 1 and DQId is equal to 0 (nal_unit_type is not equal to 20), numSubMbPart is set equal to 4.

   – Otherwise, if isDirectFlag is equal to 1 (and DQId is greater than 0 and nal_unit_type is equal to 20), numSubMbPart is set equal to 1.

   – Otherwise (isDirectFlag is equal to 0), numSubMbPart is set equal to NumSubMbPart( subMbType[ CurrMbAddr ][ mbPartIdx ] ).

2. The sub-macroblock partition index subMbPartIdx proceeds over values 0..(numSubMbPart – 1), and for each value of subMbPartIdx the SVC derivation process for luma motion vector components and reference indices of a macroblock or sub-macroblock partition as specified in subclause G.8.4.1.1 is invoked with mbPartIdx, subMbPartIdx, isDirectFlag, sliceIdc, fieldMbFlag, mbType, subMbType, predFlagL0, predFlagL1, refIdxL0,

refIdxL1, mvL0, mvL1, mvCnt, refIdxILPredL0, refIdxILPredL1, mvILPredL0, mvILPredL1, and, when DQId is equal to 0 and (slice_type % 5) is equal to 1, the reference picture list refPicList1 as the inputs and the outputs are modified versions of the arrays predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and mvCnt.

#### G.8.4.1.1 SVC derivation process for luma motion vector components and reference indices of a macroblock or sub-macroblock partition

This subclause is only invoked when mbType[ CurrMbAddr ], which is specified as input to this subclause, is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL.

Inputs to this process are

- a variable mbPartIdx specifying the current macroblock partition,
- a variable subMbPartIdx specifying the current sub-macroblock partition,
- a variable isDirectFlag specifying whether the current macroblock partition is coded in direct mode,
- a one-dimensional array sliceIdc with PicSizeInMbs elements specifying slice identifications for the macroblocks of the current layer representation,
- a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,
- a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,
- a (PicSizeInMbs)x4 array subMbType specifying sub-macroblock types for the macroblocks of the current layer representation,
- two (PicSizeInMbs)x4 arrays predFlagL0 and predFlagL1 specifying prediction utilization flags for the macroblocks of the current layer representation,
- two (PicSizeInMbs)x4 arrays refIdxL0 and refIdxL1 specifying reference indices for the macroblocks of the current layer representation,
- two (PicSizeInMbs)x4x4x2 arrays mvL0 and mvL1 specifying motion vector components for the macroblocks of the current layer representation,
- a one-dimensional array mvCnt with PicSizeInMbs elements specifying the number of motion vectors for the macroblocks of the current layer representation,
- two 2x2 arrays refIdxILPredL0 and refIdxILPredL1 specifying inter-layer reference index predictors for the current macroblock,
- two 4x4x2 arrays mvILPredL0 and mvILPredL1 specifying inter-layer motion vector predictors for the current macroblock,
- when DQId is equal to 0 and (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Outputs of this process are

- modified versions of the arrays predFlagL0 and predFlagL1,
- modified versions of the arrays refIdxL0 and refIdxL1,
- modified versions of the arrays mvL0 and mvL1,
- a modified version of the array mvCnt.

For all processes specified in clauses 6 or 8 that are invoked from the process specified in this subclause or a child process of the process specified in this subclause, the following modifications apply.

a) In subclauses 6.4.11.2 and 8.4.1.3.2, a macroblock with address mbAddr is treated as field macroblock when fieldMbFlag[ mbAddr ] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[ mbAddr ] is equal to 0. In particular, the current macroblock is treated as field macroblock when fieldMbFlag[ CurrMbAddr ] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[ CurrMbAddr ] is equal to 0.

b) In subclause 6.4.7, a macroblock with address mbAddr is treated to belong to a different slice than the current macroblock CurrMbAddr, when sliceIdc[ mbAddr ] is not equal to sliceIdc[ CurrMbAddr ].

c) In subclause 6.4.11.2, a macroblock mbAddr is treated as top macroblock when (mbAddr % 2) is equal to 0, and it is treated as bottom macroblock when (mbAddr % 2) is equal to 1.

d) In subclauses 6.4.2.1, 6.4.2.2, 6.4.10.7, 8.4.1.1, 8.4.1.3, any occurrence of mb_type is replaced by mbType[ CurrMbAddr ] with mbType being the array mbType that is input to this subclause.

e) In subclauses 6.4.2.2 and 6.4.10.7, any occurrence of sub_mb_type is replaced by subMbType[ CurrMbAddr ] with subMbType being the array subMbType that is input to this subclause.

f) In subclause 6.4.10.7, the macroblock partitioning of the macroblock mbAddrN that is used for determining the macroblock partition index mbPartIdxN and the sub-macroblock partition index subMbPartN based on a luma location ( xW, yW ) is the macroblock partitioning given by mbType[ mbAddrN ] and the specification in subclause G.8.4.1.

g) In subclause 6.4.10.7, a macroblock partition or sub-macroblock partition given by mbAddrN, mbPartIdxN, and subMbPartIdxN is treated as not yet decoded when mbAddrN is equal to CurrMbAddr and (4 * mbPartIdxN + subMbPartIdxN) is greater than (4 * mbPartIdx + subMbPartIdx).

h) In subclause 8.4.1.3.2, a macroblock with mbAddrN is treated as coded in Intra prediction mode when mbType[ mbAddrN ] is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL.

i) In subclause 8.4.1.3.2, the variable predFlagLX of a macroblock or sub-macroblock partition given by mbAddrN\mbPartIdxN\subMbPartIdxN is replaced by predFlagLX[ mbAddrN ][ mbPartIdxN ] with predFlagLX being the array predFlagLX that is input to this subclause.

j) In subclause 8.4.1.3.2, the motion vector MvLX[ mbPartIdxN ][ subMbPartIdxN ] and the reference index RefIdxLX[ mbPartIdxN ] of a macroblock or sub-macroblock partition given by mbAddrN\mbPartIdxN\subMbPartIdxN are replaced by mvLX[ mbAddrN ][ mbPartIdxN ][ subMbPartIdxN ] and refIdxLX[ mbAddrN ][ mbPartIdxN ], respectively, with mvLX and refIdxLX being the arrays mvLX and refIdxLX, respectively, that are input to this subclause.

k) In subclause 8.4.1.2.1, any occurrence of RefPicList1[ 0 ] is replaced by refPicList1[ 0 ] with refPicList1[ 0 ] being the first layer field (when field_pic_flag is equal to 1) or the first layer frame or layer complementary field pair (when field_pic_flag is equal to 0) in the reference picture list refPicList1 that is specified as input to this subclause. The reference picture list refPicList1 is a reference list of layer pictures that correspond to layer representations with DQId equal to 0 of previously decoded access units.

l) In subclause 8.4.1.2.1, the current picture CurrPic represents the current layer picture with DQId equal to 0 and variable colPic specifies the layer picture for the layer representation with DQId equal to 0 of a previously decoded access unit.

m) In subclause 8.4.1.2.1, all picture order count values are picture order count value for the dependency representation with dependency_id equal to 0.

n) In subclause 8.4.1.2.1, the modification b) specified in subclause G.8.2 applies with currDependencyId being equal to 0.

o) In subclause 8.4.1.2.1, for deriving the variable fieldDecodingFlagX, the macroblock mbAddrX is treated as field macroblock when fieldMbColPicFlag[ mbAddrX ] is equal to 1, it is treated as frame macroblock when fieldMbColPicFlag[ mbAddrX ] is equal to 0. The array fieldMbColPicFlag specifies the array fieldMbFlag that was derived by the process in subclause G.8.1.5.1 for the layer representation with DQId equal to 0.

p) In subclause 8.4.1.2.1, the variables PredFlagL0, PredFlagL1, RefIdxL0, RefIdxL1, MvL0, and MvL1 for the macroblock mbAddrCol inside the picture colPic are replaced with the predFlagL0[ mbAddrCol ], predFlagL1[ mbAddrCol ], refIdxL0[ mbAddrCol ], refIdxL1[ mbAddrCol ], mvL0[ mbAddrCol ], and mvL1[ mbAddrCol ], respectively, that have been derived for the layer picture colPic that is associated with DQId equal to 0.

q) In subclause 8.4.1.2.1, the macroblock mbAddrCol is interpreted as coded in Intra macroblock prediction mode when mbType[ mbAddrCol ] that has been derived when for the layer picture colPic that is associated with DQId equal to 0 is equal to I_16x16, I_8x8, I_4x4, or I_PCM.

r) In subclause 8.4.1.2.2, the co-located macroblock is treated as field macroblock when fieldMbColPicFlag[ mbAddrCol ] is equal to 1, it is treated as frame macroblock when fieldMbColPicFlag[ mbAddrCol ] is equal to 0. The array fieldMbColPicFlag specifies the array fieldMbFlag that was derived by the process in subclause G.8.1.5.1 for the layer representation with DQId equal to 0. The macroblock address mbAddrCol is the macroblock address of the co-located macroblock as derived in subclause 8.4.1.2.1.

The reference index predictors refIdxPredL0 and refIdxPredL1, the motion vector predictors mvPredL0 and mvPredL1, and the variable mvCntInc are derived as follows.

– If mbType[ CurrMbAddr ] is equal to P_Skip, the reference index predictor refIdxPredL1 is set equal to –1, both components of the motion vector predictor mvPredL1 are set equal to 0, the variable mvCntInc is set equal to 1, and the derivation process for luma motion vectors for skipped macroblocks in P slices as specified in subclause 8.4.1.1 is invoked with the outputs being assigned to the motion vector predictor mvPredL0 and the reference index predictor refIdxPredL0. For this invocation of the process in subclause 8.4.1.1, the modifications specified above in items a) through j) of this subclause apply.

– Otherwise, if isDirectFlag is equal to 1 and DQId is equal to 0 (nal_unit_type is not equal to 20), the derivation process for spatial direct luma motion vector and reference index prediction mode as specified in subclause 8.4.1.2.2 is invoked with mbPartIdx and subMbPartIdx as the inputs and the output variables refIdxL0, refIdxL1, mvL0, mvL1, and subMvCnt are assigned to the reference index predictors refIdxPredL0 and refIdxPredL1, the motion vectors predictors mvPredL0 and mvPredL1, and the variable subMvCnt, respectively. For this invocation of the process in subclause 8.4.1.2.2, the modifications specified above in items a) through r) of this subclause apply.

NOTE – When the current subclause is invoked, direct_spatial_mv_pred_flag is always equal to 1.

– Otherwise, if isDirectFlag is equal to 1 (and DQId is greater than 0 and nal_unit_type is equal to 20), the SVC derivation process for luma motion vectors and reference indices for B_Skip, B_Direct_16x16, and B_Direct_8x8 in NAL units with nal_unit_type equal to 20 as specified in subclause G.8.4.1.2 is invoked with mbPartIdx, fieldMbFlag, mbType, subMbType, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, and mvL1 as the inputs and the outputs are refIdxPredL0, refIdxPredL1, mvPredL0, mvPredL1, and mvCntInc.

– Otherwise, the variable mvCntInc is initially set equal to 0, and for X being replaced by 0 and 1, the following applies.

  – If any of the following conditions is true, refIdxPredLX is set equal to –1 and both components of mvPredLX are set equal to 0.

    – mbType[ CurrMbAddr ] is not equal to P_8x8, P_8x8ref0, or B_8x8 and MbPartPredMode( mbType[ CurrMbAddr ], mbPartIdx ) is not equal to Pred_LX or BiPred

    – mbType[ CurrMbAddr ] is equal to P_8x8, P_8x8ref0, or B_8x8 and SubMbPartPredMode( subMbType[ CurrMbAddr ][ mbPartIdx ] ) is not equal to Pred_LX or BiPred

  – Otherwise, if base_mode_flag is equal to 1 or motion_prediction_flag_lX[ mbPartIdx ] is equal to 1, the following ordered steps are specified.

    1. The inverse macroblock partition scanning process as specified in subclause 6.4.2.1 is invoked with mbPartIdx as the input and the output is assigned to ( xP, yP ). For this invocation of the process in subclause 6.4.2.1, the modification specified above in item d) of this subclause applies.

    2. Inverse sub-macroblock partition scanning process as specified in subclause 6.4.2.2 is invoked with mbPartIdx and subMbPartIdx as the inputs and the output is assigned to ( xS, yS ). For this invocation of the process in subclause 6.4.2.2, the modifications specified above in items d) and e) of this subclause apply.

    3. The reference index predictor refIdxPredLX and the motion vector predictor mvPredLX are derived by

$$\begin{aligned} \text{refIdxPredLX} &= \text{refIdxILPredLX}[\,(xP + xS)/8,\,(xP + xS)/8\,] \\ \text{mvPredLX}[\,c\,] &= \text{mvILPredLX}[\,(xP + xS)/4,\,(xP + xS)/4\,][\,c\,] \quad \text{with } c = 0..1 \end{aligned} \tag{G-106}$$

       The bitstream shall not contain data that result in refIdxPredLX less than 0.

    4. mvCntInc is set equal to (mvCntInc + 1).

  – Otherwise, the following ordered steps are specified.

    1. Depending on mbType[ CurrMbAddr ], the reference index predictor refIdxPredLX is derived as follows.

       – If mbType[ CurrMbAddr ] is equal to P_8x8ref0, refIdxPredLX is set equal to 0.

       – Otherwise (mbType[ CurrMbAddr ] is not equal to P_8x8ref0), refIdxPredLX is set equal to ref_idx_lX[ mbPartIdx ].

    2. The derivation process for luma motion vector prediction as specified in subclause 8.4.1.3 is invoked with mbPartIdx, subMbPartIdx, refIdxPredLX, and currSubMbType set equal to

subMbType[ CurrMbAddr ][ mbPartIdx ] as the inputs and the output is assigned to mvPredLX. For this invocation of the process in subclause 8.4.1.3, the modifications specified in items a) through j) of this subclause apply

3. mvCntInc is set equal to (mvCntInc + 1).

For X being replaced by 0 and 1, the arrays refIdxLX, predFlagLX, and mvLX are modified by applying the following ordered steps.

1. When subMbPartIdx is equal to 0, the arrays refIdxLX and predFlagLX are modified by

$$refIdxLX[\ CurrMbAddr\ ][\ mbPartIdx\ ] = refIdxPredLX \quad \text{(G-107)}$$
$$predFlagLX[\ CurrMbAddr\ ][\ mbPartIdx\ ] = (\ refIdxPredLX < 0\ ?\ 0\ :\ 1\ ) \quad \text{(G-108)}$$

2. The array mvLX is modified by.

$$mvLX[\ CurrMbAddr\ ][\ mbPartIdx\ ][\ subMbPartIdx\ ][\ c\ ] = mvPredLX[\ c\ ] \quad \text{with } c = 0..1 \quad \text{(G-109)}$$

3. When predFlagLX[ CurrMbAddr ][ mbPartIdx ] is equal to 1, base_mode_flag is equal to 0, isDirectFlag is equal to 0, and mbType[ CurrMbAddr ] is not equal to P_Skip, the array mvLX is modified by

$$mvLX[\ CurrMbAddr\ ][\ mbPartIdx\ ][\ subMbPartIdx\ ][\ c\ ] \mathrel{+}= mvd\_lX[\ mbPartIdx\ ][\ subMbPartIdx\ ][\ c\ ] \quad \text{with } c = 0..1 \quad \text{(G-110)}$$

The array mvCnt is modified as follows.

– If mbPartIdx is equal to 0 and subMbPartIdx is equal to 0, mvCnt[ CurrMbAddr ] is set equal to mvCntInc.

– Otherwise (mbPartIdx is greater than 0 or subMbPartIdx is greater than 0), the array mvCnt is modified by

$$mvCnt[\ CurrMbAddr\ ] \mathrel{+}= mvCntInc \quad \text{(G-111)}$$

**G.8.4.1.2 SVC derivation process for luma motion vectors and reference indices for B_Skip, B_Direct_16x16, and B_Direct_8x8 in NAL units with nal_unit_type equal to 20**

Inputs to this process are

– a variable mbPartIdx specifying the current macroblock partition,

– a one-dimensional array fieldMbFlag with PicSizeInMbs elements specifying which macroblocks of the current layer representation are coded as field macroblocks and which macroblocks are coded as frame macroblocks,

– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current layer representation,

– a (PicSizeInMbs)x4 array subMbType specifying sub-macroblock types for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4 arrays predFlagL0 and predFlagL1 specifying prediction utilization flags for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4 arrays refIdxL0 and refIdxL1 specifying reference indices for the macroblocks of the current layer representation,

– two (PicSizeInMbs)x4x4x2 arrays mvL0 and mvL1 specifying motion vector components for the macroblocks of the current layer representation.

Outputs of this process are

– the reference index predictors refIdxPredL0 and refIdxPredL1,

– the motion vector predictors mvPredL0 and mvPredL1,

– the variable mvCntInc.

The variable currSubMbType is derived as follows.

– If mbType[ CurrMbAddr ] is equal to B_Skip or B_Direct_16x16, currSubMbType is marked as "unspecified".

– Otherwise (mbType[ CurrMbAddr ] is equal to B_8x8 and subMbType[ CurrMbAddr ][ mbPartIdx ] is equal to B_Direct_8x8), currSubMbType is set equal to B_Bi_8x8.

NOTE – The variable currSubMbType is only used for deriving the variable predPartWidth in subclause 6.4.10.7, which specifies the partition width of the current macroblock or sub-macroblock partition for determining neighbouring partitions that are used for motion vector prediction. Inside subclause 6.4.10.7, the variable predPartWidth is set equal to 16 when the current macroblock is coded with macroblock type equal to B_Skip or B_Direct_16x16 or the current sub-macroblock is coded with sub-macroblock type equal B_Direct_8x8. When the current subclause is invoked for a sub-macroblock coded with sub-macroblock type equal to B_Direct_8x8 (the current subclause is only invoked for NAL units with nal_unit_type equal to 20), currSubMbType is set equal to B_Bi_8x8 in order to set the variable predPartWidth equal to 8 in subclause 6.4.10.7.

For X being replaced by 0 and 1, the reference index predictor refIdxPredLX is derived by applying the following ordered steps.

1. The derivation process for motion data of neighbouring partitions as specified in subclause 8.4.1.3.2 is invoked with mbPartIdx, subMbPartIdx set equal to 0, currSubMbType, and listSuffixFlag set equal to X as the inputs and the outputs are the reference indices refIdxLXN with N being replaced by A, B, and C. For this invocation of the process in subclause 8.4.1.3.2, the modifications specified in items a) through j) of subclause G.8.4.1.1 apply.
2. The reference index predictor refIdxPredLX is derived by

$$refIdxPredLX = MinPositive( refIdxLXA, MinPositive( refIdxLXB, refIdxLXC ) ) \quad \text{(G-112)}$$

with

$$MinPositive( x, y ) = \begin{cases} Min(x, y) & \text{if } x >= 0 \text{ and } y >= 0 \\ Max(x, y) & \text{otherwise} \end{cases} \quad \text{(G-113)}$$

When both reference index predictors refIdxPredL0 and refIdxPredL1 are less than 0, refIdxPredL0 and refIdxPredL1 are set equal to 0.

For X being replaced by 0 and 1, the motion vector predictor mvPredLX is derived as follows.

– If refIdxPredLX is greater than or equal to 0, the derivation process for luma motion vector prediction as specified in subclause 8.4.1.3 is invoked with mbPartIdx, subMbPartIdx set equal to 0, refIdxPredLX, and currSubMbType as the inputs and the output is assigned to mvPredLX. For this invocation of the process in subclause 8.4.1.3, the modifications specified in items a) through j) of subclause G.8.4.1.1 apply.

– Otherwise, both components of the motion vector mvPredLX are set equal to 0.

The variable mvCntInc is derived as specified by the following ordered steps.

1. mvCntInc is set equal to 0.
2. When refIdxPredL0 is greater than or equal to 0, mvCntInc is set equal to (mvCntInc + 1).
3. When refIdxPredL1 is greater than or equal to 0, mvCntInc is set equal to (mvCntInc + 1).

### G.8.4.2 SVC decoding process for Inter prediction samples

Inputs to this process are

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable mbType specifying the macroblock type for the current macroblock,

– a list subMbType with 4 elements specifying the sub-macroblock types for the current macroblock,

– two lists predFlagL0 and predFlagL1 with 4 elements specifying prediction utilization flags for the current macroblock,

– two lists refIdxL0 and refIdxL1 with 4 elements specifying reference indices for the current macroblock,

– two 4x4x2 arrays mvL0 and mvL1 specifying motion vectors components for the current macroblock,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation,

– the reference picture lists refPicList0 and refPicList1 (when available),

– a (PicWidthInSamples$_L$)x(PicHeightInSamples$_L$) array picSamples$_L$ of luma sample values,

– when ChromaArrayType is not equal to 0, two (PicWidthInSamples$_C$)x(PicHeightInSamples$_C$) arrays picSamples$_{Cb}$ and picSamples$_{Cr}$ of chroma sample values.

Outputs of this process are

– a modified version of the array of luma sample values picSamples$_L$,

– when ChromaArrayType is not equal to 0, modified versions of the two arrays of chroma sample values picSamples$_{Cb}$ and picSamples$_{Cr}$.

For all processes specified in clauses 6 or 8 that are invoked from the process specified in this subclause or a child process of the process specified in this subclause, the following modifications apply.

a) In subclauses 8.4.3, 8.4.1.4, and 8.4.2.1, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0. When field_pic_flag is equal to 0 and the current macroblock CurrMbAddr is a field macroblock, its parity is equal to top when (CurrMbAddr % 2) is equal to 0 and its parity is equal to bottom when (CurrMbAddr % 2) is equal to 1.

b) In subclauses 8.4.3 and 8.4.2.1, any occurrence of RefPicList0 or RefPicList1 is replaced with refPicList0 or refPicList1, respectively, with refPicList0 and refPicList1 being the reference picture lists specified as inputs to this subclause.

c) In subclause 8.4.1.4, the reference picture referred by refIdxLX is specified by refPicListX[ refIdxLX ] with refPicList0 and refPicList1 specified as inputs to this subclause.

d) In subclauses 8.4.2.2.1 and 8.4.2.2.2, any occurrence of mb_field_decoding_flag is replaced by fieldMbFlag.

Let predMb$_L$ be a 16x16 array of luma prediction samples for the macroblock mbAddr.

When ChromaArrayType is not equal to 0, let predMb$_{Cb}$ and predMb$_{Cr}$ be two (MbWidthC)x(MbHeightC) arrays of chroma prediction samples for the macroblock mbAddr.

The variable isDirectFlag is derived as follows.

– If any of the following conditions is true, isDirectFlag is set equal to 1.

  – mbType is equal to B_Skip or B_Direct_16x16

  – mbType is equal to B_8x8 and subMbType[ mbPartIdx ] is equal to B_Direct_8x8

– Otherwise, isDirectFlag is set equal to 0.

The variable numMbPart is derived as follows.

– If mbType is equal to B_Skip or B_Direct_16x16 and DQId is equal to 0 (nal_unit_type is not equal to 20), numMbPart is set equal to 4.

– Otherwise, if mbType is equal to B_Skip or B_Direct_16x16 (and DQId is greater than 0 and nal_unit_type is equal to 20), numMbPart is set equal to 1.

– Otherwise (mbType is not equal to B_Skip or B_Direct_16x16), numMbPart is set equal to NumMbPart( mbType ).

The macroblock partition index mbPartIdx proceeds over the values 0..(numMbPart – 1), and for each value of mbPartIdx the following ordered steps are specified.

1. The SVC derivation process for prediction weights as specified in subclause G.8.4.2.1 is invoked with fieldMbFlag, refIdxL0[ mbPartIdx ], refIdxL1[ mbPartIdx ], predFlagL0[ mbPartIdx ], predFlagL1[ mbPartIdx ], refPicList0, and refPicList1 (when available) as inputs and the outputs are assigned to logWD$_L$, w$_{0L}$, w$_{1L}$, o$_{0L}$, o$_{1L}$, and when ChromaArrayType is not equal to 0, logWD$_C$, w$_{0C}$, w$_{1C}$, o$_{0C}$, o$_{1C}$ with C being replaced by Cb and Cr.

2. The luma location ( xP, yP ) is derived as follows.

– If mbType is equal to B_Skip or B_Direct_16x16, xP is set equal to (8 * ( mbPartIdx % 2 )) and yP is set equal to (8 * ( mbPartIdx / 2 )).

– Otherwise (mbType is not equal to B_Skip or B_Direct_16x16), the inverse macroblock partition scanning process as specified in subclause 6.4.2.1 is invoked with mbPartIdx as the input and the output is assigned to ( xP, yP ). For this invocation of the process in subclause 6.4.2.1, any occurrence of mb_type is replaced by mbType.

3. The variable numSubMbPart is derived as follows.

– If isDirectFlag is equal to 1 and DQId is equal to 0 (nal_unit_type is not equal to 20), numSubMbPart is set equal to 4.

– Otherwise, if isDirectFlag is equal to 1 (and DQId is greater than 0 and nal_unit_type is equal to 20), numSubMbPart is set equal to 1.

– Otherwise (isDirectFlag is equal to 0), numSubMbPart is set equal to NumSubMbPart( subMbType[ mbPartIdx ] )

4. The sub-macroblock partition index proceeds over values 0..(numSubMbPart – 1), and for each value of subMbPartIdx the following ordered steps are specified.

a. The variables partWidth and partHeight are derived as follows.

– If isDirectFlag is equal to 1 and DQId is equal to 0 (nal_unit_type is not equal to 20), partWidth and partHeight are set equal to 4.

– Otherwise, if isDirectFlag is equal to 1 (and DQId is greater than 0 and nal_unit_type is equal to 20), the following applies.

– If mbType is equal to B_Skip or B_Direct_16x16, partWidth and partHeight are set equal to 16.

– Otherwise (mbType is equal to B_8x8 and subMbType[ mbPartIdx ] is equal to B_Direct_8x8), partWidth and partHeight are set equal to 8.

– Otherwise (isDirectFlag is equal to 0), the following applies.

– If mbType is not equal to P_8x8, P_8x8ref0, or B_8x8, partWidth and partHeight are derived by

partWidth = MbPartWidth( mbType ) (G-114)
partHeight = MbPartHeight( mbType ) (G-115)

– Otherwise (mbType is equal to P_8x8, P_8x8ref0, or B_8x8), partWidth and partHeight are derived by

partWidth = SubMbPartWidth( subMbType[ mbPartIdx ] ) (G-116)
partHeight = SubMbPartHeight( subMbType[ mbPartIdx ] ) (G-117)

b. When ChromaArrayType is not equal to 0, the variables partWidthC and partHeightC are derived by

partWidthC = partWidth / SubWidthC (G-118)
partHeightC = partHeight / SubWidthC (G-119)

c. For X being replaced by 0 and 1, when ChromaArrayType is not equal to 0 and predFlagLX[ mbPartIdx ] is equal to 1, the derivation process for chroma motion vectors as specified in subclause 8.4.1.4 is invoked with mvLX[ mbPartIdx ][ subMbPartIdx ] and refIdxLX[ mbPartIdx ] as the inputs and the output is the chroma motion vector mvCLX. For this invocation of the process in subclause 8.4.1.4, the modifications specified above in items a) and c) of this subclause apply.

d. The decoding process for Inter prediction samples as specified in subclause 8.4.2 is invoked with mbPartIdx, subMbPartIdx, partWidth and partHeight, partWidthC and partHeightC (if available), luma motion vectors mvL0[ mbPartIdx ][ subMbPartIdx ] and mvL1[ mbPartIdx ][ subMbPartIdx ], chroma motion vectors mvCL0 and mvCL1 (if available), reference indices refIdxL0[ mbPartIdx ] and refIdxL1[ mbPartIdx ], prediction utilization flags predFlagL0[ mbPartIdx ] and predFlagL1[ mbPartIdx ] as well as variables for weighted prediction $\log WD_L$, $w_{0L}$, $w_{1L}$, $o_{1L}$, $o_{0L}$, and when ChromaArrayType is not equal to 0, $\log WD_C$, $w_{0C}$, $w_{1C}$, $o_{1C}$, and $o_{0C}$ (with C being replaced by Cb and Cr) as the inputs and the outputs are a (partWidth)x(partHeight) array $predPart_L$ of luma prediction samples and, when ChromaArrayType is not equal to 0, two (partWidthC)x(partHeightC) arrays $predPart_{Cb}$ and $predPart_{Cr}$ of chroma prediction samples. For this invocation of the process in subclause 8.4.2, the modifications specified above in items a), b), and d) of this subclause apply.

e. The luma location ( xS, yS ) is derived as follows.

- If mbType is not equal to B_8x8 or subMbType[ mbPartIdx ] is equal to B_Direct_8x8, xS is set equal to (4 * ( subMbPartIdx % 2 )) and yP is set equal to (4 * ( subMbPartIdx / 2 )).
- Otherwise (mbType is equal to B_8x8 and subMbType[ mbPartIdx ] is not equal to B_Direct_8x8), the inverse sub-macroblock partition scanning process as specified in subclause 6.4.2.2 is invoked with mbPartIdx and subMbPartIdx as the inputs and the output is assigned to ( xS, yS ). For this invocation of the process in subclause 6.4.2.2, any occurrence of mb_type is replaced by mbType and any occurrence of sub_mb_type is replaced by subMbType.

f. For x = 0..(partWidth – 1) and y = 0..(partHeight – 1), the 16x16 array $predMb_L$ is modified by

$$predMb_L[ xP + xS + x, yP + yS + y ] = predPart_L[ x, y ] \quad \text{(G-120)}$$

g. When ChromaArrayType is not equal to 0, for x = 0..(partWidthC – 1) and y = 0..(partHeightC – 1), the (MbWidthC)x(MbHeightC) arrays $predMb_{Cb}$ and $predMb_{Cr}$ are modified by

$$predMb_{Cb}[ ( xP + xS ) / SubWidthC + x, ( yP + yS ) / SubHeightC + y ] = predPart_{Cb}[ x, y ] \quad \text{(G-121)}$$
$$predMb_{Cr}[ ( xP + xS ) / SubWidthC + x, ( yP + yS ) / SubHeightC + y ] = predPart_{Cr}[ x, y ] \quad \text{(G-122)}$$

When base_mode_flag is equal to 1, quality_id is equal to 0, MbaffFrameFlag is equal to 0, RefLayerMbaffFrameFlag is equal to 0, and RestrictedSpatialResolutionChangeFlag is equal to 0, the intra-inter prediction combination process specified in subclause G.8.4.2.2 is invoked with fieldMbFlag, refLayerFieldMbFlag, refLayerMbType, $predMb_L$, $picSamples_L$, and, when ChromaArrayType is not equal to 0, $predMb_{Cb}$, $predMb_{Cr}$, $picSamples_{Cb}$, and $picSamples_{Cr}$ as the inputs, and the outputs are a modified version of $predMb_L$ and, when ChromaArrayType is not equal to 0, modified versions of $predMb_{Cb}$ and $predMb_{Cr}$.

The picture sample array construction process as specified in subclause G.8.5.4.1 is invoked with fieldMbFlag, $predMb_L$, $picSamples_L$, and, when ChromaArrayType is not equal to 0, $predMb_{Cb}$, $predMb_{Cr}$, $picSamples_{Cb}$, and $picSamples_{Cr}$ as the inputs and the outputs are a modified version of $picSamples_L$ and, when ChromaArrayType is not equal to 0, modified versions of $picSamples_{Cb}$, and $picSamples_{Cr}$.

**G.8.4.2.1 SVC derivation process for prediction weights**

Inputs to this process are

- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- the reference indices refIdxL0 and refIdxL1 for the current macroblock partition,
- the prediction list utilization flags predFlagL0 and predFlagL1 for the current macroblock partition,
- the reference picture lists refPicList0 and refPicList1 (when available).

Outputs of this process are

- variables for weighted prediction of luma samples $logWD_L$, $w_{0L}$, $w_{1L}$, $o_{0L}$, $o_{1L}$.
- when ChromaArrayType is not equal to 0 (monochrome), variables for weighted prediction of chroma samples $logWD_C$, $w_{0C}$, $w_{1C}$, $o_{0C}$, $o_{1C}$ with C being replaced by Cb and Cr.

Depending on base_pred_weight_table_flag the following applies.

- If base_pred_weight_table_flag is equal to 0, the derivation process for prediction weights as specified in subclause 8.4.3 is invoked with refIdxL0, refIdxL1, predFlagL0, and predFlagL1 as inputs and the outputs are assigned to $logWD_L$, $w_{0L}$, $w_{1L}$, $o_{0L}$, $o_{1L}$, and when ChromaArrayType is not equal to 0, $logWD_C$, $w_{0C}$, $w_{1C}$, $o_{0C}$, $o_{1C}$ with C being replaced by Cb and Cr. For this invocation of the process in subclause 8.4.3, the modifications specified in items a) and b) of subclause G.8.4.2 apply.
- Otherwise (base_pred_weight_table_flag is equal to 1), for X being replaced by 0 and 1, the following ordered steps are specified.
  1. Let baseSlice be any slice of the layer representation with DQId equal to the value of ref_layer_dq_id.
  2. Let refLayerLumaLogWD, aRefLayerLumaWeightLX[], and aRefLayerLumaOffsetLX[] be variables that are set equal to the values of the syntax elements luma_log2_weight_denom, luma_weight_lX[], and luma_offset_lX[], respectively, of baseSlice.

3. When ChromaArrayType is not equal to 0, let refLayerChromaLogWD, aRefLayerChromaWeightLX[][], and aRefLayerChromaOffsetLX[][] be variables that are set equal to the values of the syntax elements chroma_log2_weight_denom, chroma_weight_lX[], and chroma_offset_lX[], respectively, of baseSlice.

4. The variable refIdxLXWP is derived as follows

   – If MbaffFrame is equal to 1 and fieldMbFlag is equal to 1,

   $$refIdxLXWP = refIdxLX >> 1 \quad \text{(G-123)}$$

   – Otherwise (MbaffFrameFlag is equal to 0 or fieldMbFlag is equal to 0),

   $$refIdxLXWP = refIdxLX \quad \text{(G-124)}$$

5. The variables $logWD_Y$, $w_{XY}$, $o_{XY}$ are derived by

$$logWD_Y = baseLumaLogWD \quad \text{(G-125)}$$
$$w_{XY} = aRefLayerLumaWeightLX[\ refIdxLXWP\ ] \quad \text{(G-126)}$$
$$o_{XY} = aRefLayerLumaOffsetLX[\ refIdxLXWP\ ] * (\ 1 << (\ BitDepth_Y - 8\ )\ ) \quad \text{(G-127)}$$

6. When ChromaArrayType is not equal to 0, the variables $logWD_C$, $w_{XC}$, $o_{XC}$ (with C being replaced by Cb and Cr and iCbCr = 0 for Cb and iCbCr = 1 for Cr) are derived by

$$logWD_C = baseChromaLogWD \quad \text{(G-128)}$$
$$w_{XC} = aRefLayerChromaWeightLX[\ refIdxLXWP\ ][\ iCbCr\ ] \quad \text{(G-129)}$$
$$o_{XC} = aRefLayerChromaOffsetLX[\ refIdxLXWP\ ][\ iCbCr\ ] * (\ 1 << (\ BitDepth_C - 8\ )\ ) \quad \text{(G-130)}$$

7. When base_pred_weight_table_flag is equal to 1 and predFlagL0 and predFlagL1 are equal to 1, the following constraint shall be obeyed for C equal to L and, when ChromaArrayType is not equal to 0, Cb and Cr

$$-128 <= w_{0C} + w_{1C} <= (\ (\ logWD_C == 7\ )\ ?\ 127 : 128\ ) \quad \text{(G-131)}$$

**G.8.4.2.2 Intra-inter prediction combination process**

Inputs to this process are

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation,

– a 16x16 array $predMb_L$ of luma inter prediction samples for the current macroblock,

– a (PicWidthInSamples$_L$)x(PicHeightInSamples$_L$) array $picSamples_L$ of luma sample values,

– when ChromaArrayType is not equal to 0, two (MbWidthC)x(MbHeightC) arrays $predMb_{Cb}$ and $predMb_{Cr}$ of chroma prediction samples for the macroblock mbAddr,

– when ChromaArrayType is not equal to 0, two (PicWidthInSamples$_C$)x(PicHeightInSamples$_C$) arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ of chroma sample values.

Outputs of this process are

– a modified version of the array $predMb_L$ of luma prediction samples for the macroblock mbAddr,

– when ChromaArrayType is not equal to 0, modified versions of the two arrays $predMb_{Cb}$ and $predMb_{Cr}$ of chroma prediction samples for the macroblock mbAddr.

Let $predMbTemp_L$ be a 16x16 array and, when ChromaArrayType is not equal to 0, let $predMbTemp_{Cb}$ and $predMbTemp_{Cr}$ be two (MbWidthC)x(MbHeightC) arrays. The macroblock sample array extraction process as specified in subclause G.8.5.4.2 is invoked with fieldMbFlag, $picSamples_L$, and when ChromaArrayType is not equal to 0,

$picSamples_{Cb}$ and $picSamples_{Cr}$ as the inputs and the outputs are assigned to $predMbTemp_L$, and when ChromaArrayType is not equal to 0, $predMbTemp_{Cb}$ and $predMbTemp_{Cr}$.

For x proceeding over the values 0..15 and y proceeding over the values 0..15, the following ordered steps are specified.

1. The derivation process for reference layer macroblocks as specified in subclause G.6.1 is invoked with ( x, y ), fieldMbFlag, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are assigned to mbAddrRefLayer and ( xRef, yRef ).
2. When refLayerMbType[ mbAddrRefLayer ] is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, the following applies.
   a. The prediction luma sample $predMb_L[ x, y ]$ is modified by

$$predMb_L[ x, y ] = predMbTemp_L[ x, y ] \quad \text{(G-132)}$$

   b. When ChromaArrayType is not equal to 0, (x % SubWidthC) is equal to 0, and (y % SubHeightC) is equal to 0, the prediction chroma sample $predMb_{Cb}[ x, y ]$ and $predMb_{Cr}[ x, y ]$ are modified by

$$predMb_{Cb}[ x, y ] = predMbTemp_{Cb}[ x, y ] \quad \text{(G-133)}$$
$$predMb_{Cr}[ x, y ] = predMbTemp_{Cr}[ x, y ] \quad \text{(G-134)}$$

### G.8.5 SVC transform coefficient decoding and sample array construction processes

Subclause G.8.5.1 specifies the transform coefficient scaling and refinement process.

Subclause G.8.5.2 specifies the transform coefficient level scaling process prior to transform coefficient refinement.

Subclause G.8.5.3 specifies the residual construction and accumulation process.

Subclause G.8.5.4 specifies the sample array accumulation process.

Subclause G.8.5.5 specifies the sample array re-initialisation process.

#### G.8.5.1 Transform coefficient scaling and refinement process

Inputs to this process are

- a variable refinementFlag specifying whether the transform coefficients for the current macroblock are combined with the existent transform coefficients for the current macroblock, which were obtained from the reference layer representation,
- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a variable cTrafo specifying the transform type,
- a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,
- a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are

- a modified version of the list sTCoeff,
- a modified version of the list tCoeffLevel.

The scaling functions and, when ChromaArrayType is not equal to 0, the chroma quantisation parameters $QP'_{Cb}$ and $QP'_{Cr}$ are derived by invoking the derivation process for chroma quantisation parameters and scaling function as specified in subclause 8.5.8.

When refinementFlag is equal to 0, all (256 + 2 * MbWidthC * MbHeightC) elements of the lists sTCoeff and tCoeffLevel are set equal to 0.

The refinement process for luma transform coefficients as specified in subclause G.8.5.1.1 is invoked with fieldMbFlag, cTrafo, sTCoeff, and tCoeffLevel as the inputs and the outputs are modified versions of the lists sTCoeff and tCoeffLevel.

When ChromaArrayType is not equal to 0, the refinement process for chroma transform coefficients as specified in subclause G.8.5.1.2 is invoked with fieldMbFlag, cTrafo, sTCoeff, and tCoeffLevel as the inputs and the outputs are modified versions of the lists sTCoeff and tCoeffLevel.

#### G.8.5.1.1 Refinement process for luma transform coefficients or chroma transform coefficients with ChromaArrayType equal to 3

Inputs to this process are

- a variable iYCbCr specifying the colour component (when present),
- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a variable cTrafo specifying the transform type,
- a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,
- a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are

- a modified version of the list sTCoeff,
- a modified version of the list tCoeffLevel.

When iYCbCr is not present as input to this subclause, it is inferred to be equal to 0.

Depending on iYCbCr, the variables bitDepth, qP, cO, coeffLevel, coeffLevel8x8, coeffDCLevel, and coeffACLevel are derived as follows.

- If iYCbCr is equal to 0, bitDepth is set equal to $BitDepth_Y$, qP is set equal to $QP'_Y$, cO is set equal to 0, coeffLevel is set equal to LumaLevel, coeffLevel8x8 is set equal to LumaLevel8x8, coeffDCLevel is set equal to Intra16x16DCLevel, and coeffACLevel is set equal to Intra16x16ACLevel.
- Otherwise, if iYCbCr is equal to 1, bitDepth is set equal to $BitDepth_C$, qP is set equal to $QP'_{Cb}$, cO is set equal to 256, coeffLevel is set equal to CbLevel, coeffLevel8x8 is set equal to CbLevel8x8, coeffDCLevel is set equal to CbIntra16x16DCLevel, and coeffACLevel is set equal to CbIntra16x16ACLevel.
- Otherwise (iYCbCr is equal to 2), bitDepth is set equal to $BitDepth_C$, qP is set equal to $QP'_{Cr}$, cO is set equal to (256 + MbWidthC * MbHeightC), coeffLevel is set equal to CrLevel, coeffLevel8x8 is set equal to CrLevel8x8, coeffDCLevel is set equal to CrIntra16x16DCLevel, and coeffACLevel is set equal to CrIntra16x16ACLevel.

Depending on cTrafo, the following applies.

- If cTrafo is equal to T_PCM, the assignment process for luma transform coefficient values or chroma transform coefficient values with ChromaArrayType equal to 3 for I_PCM macroblocks as specified in subclause G.8.5.1.1.1 is invoked with iYCbCr, sTCoeff, and tCoeffLevel as the inputs and the outputs are modified versions of sTCoeff and tCoeffLevel.
- Otherwise, if cTrafo is equal to T_4x4, the refinement process for transform coefficients of residual 4x4 blocks as specified in subclause G.8.5.1.1.2 is invoked with fieldMbFlag, bitDepth, qP, cO, coeffLevel, sTCoeff, and tCoeffLevel as the inputs and the outputs are modified versions of the lists sTCoeff and tCoeffLevel.
- Otherwise, if cTrafo is equal to T_8x8, the refinement process for transform coefficients of residual 8x8 blocks as specified in subclause G.8.5.1.1.3 is invoked with fieldMbFlag, bitDepth, qP, cO, coeffLevel8x8, sTCoeff, and tCoeffLevel as the inputs and the outputs are modified versions of the lists sTCoeff and tCoeffLevel.
- Otherwise (cTrafo is equal to T_16x16), the refinement process for transform coefficients of Intra_16x16 macroblocks as specified in subclause G.8.5.1.1.4 is invoked with fieldMbFlag, bitDepth, qP, cO, coeffDCLevel, coeffACLevel, coeffLevel8x8, sTCoeff, and tCoeffLevel as the inputs and the outputs are modified versions of the lists sTCoeff and tCoeffLevel.

##### G.8.5.1.1.1 Assignment process for luma transform coefficient values or chroma transform coefficient values with ChromaArrayType equal to 3 for I_PCM macroblocks

Inputs to this process are

- a variable iYCbCr specifying the colour component,
- a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,
- a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are

- a modified version of the list sTCoeff,
- a modified version of the list tCoeffLevel.

Depending on iYCbCr, the variables cO, cListOffset and pcmSample are derived by

```
cO          =  iYCbCr * 256
cListOffset = ( iYCbCr == 0 ? 0 : ( iYCbCr – 1 ) * 256 )                    (G-135)
pcmSample = ( iYCbCr == 0 ? pcm_luma_samples : pcm_chroma_samples )         (G-136)
```

When base_mode_flag is equal to 0, the lists tCoeffLevel and sTCoeff are modified by

$$tCoeffLevel[ cO + k ] = 0 \quad \text{with } k = 0..255 \tag{G-137}$$

$$sTCoeff[ cO + k ] = pcmSample[ cListOffset + k ] \quad \text{with } k = 0..255 \tag{G-138}$$

#### G.8.5.1.1.2 Refinement process for transform coefficients of residual 4x4 blocks

Inputs to this process are

- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a variable bitDepth specifying the bit depth,
- a variable qP specifying the quantisation parameter value,
- a variable cO specifying the first coefficient index in the list of scaled transform coefficient values sTCoeff and in the list of transform coefficient values tCoeffLevel,
- a variable coeffLevel representing LumaLevel, CbLevel, or CrLevel,
- a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,
- a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are

- a modified version of the list sTCoeff,
- a modified version of the list tCoeffLevel.

Depending on tcoeff_level_prediction_flag, the following applies.

- If tcoeff_level_prediction_flag is equal to 1, the list sTCoeff is modified by

$$sTCoeff[ cO + k ] = 0 \quad \text{with } k = 0..255 \tag{G-139}$$

- Otherwise (tcoeff_level_prediction_flag is equal to 0), the list tCoeffLevel is modified by

$$tCoeffLevel[ cO + k ] = 0 \quad \text{with } k = 0..255 \tag{G-140}$$

For each residual 4x4 block indexed by c4x4BlkIdx = 0..15, the following ordered steps are specified.

1. The inverse scanning process for transform coefficients as specified in subclause 8.5.6 is invoked with coeffLevel[ c4x4BlkIdx ] as the input and the outputs are transform coefficient level values as a 4x4 array c with elements $c_{ij}$. For this invocation of the process in subclause 8.5.6, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.
2. The list tCoeffLevel and the 4x4 array c are modified by

$$tCoeffLevel[ cO + 16 * c4x4BlkIdx + 4 * i + j ] \mathrel{+}= c_{ij} \quad \text{with } i, j = 0..3 \tag{G-141}$$

$$c_{ij} = tCoeffLevel[ cO + 16 * c4x4BlkIdx + 4 * i + j ] \quad \text{with } i, j = 0..3 \tag{G-142}$$

3. The scaling process for residual 4x4 blocks as specified in subclause 8.5.11.1 is invoked with bitDepth, qP, and the 4x4 array c as the inputs and the outputs are scaled transform coefficient values as a 4x4 array d with elements $d_{ij}$. For this invocation of the process in subclause 8.5.11.1, the array c is treated as not relating to a luma residual block coded using the Intra_16x16 prediction mode and as not relating to a chroma residual block.
4. The list sTCoeff is modified by

$$sTCoeff[ cO + 16 * c4x4BlkIdx + 4 * i + j ] \mathrel{+}= d_{ij} \quad \text{with } i, j = 0..3 \tag{G-143}$$

The bitstream shall not contain data that result in any element sTCoeff[ cO + k ] with k = 0..255 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)}-1$, inclusive.

The bitstream shall not contain data that result in any element tCoeffLevel[ cO + k ] with k = 0..255 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)}-1$, inclusive.

#### G.8.5.1.1.3 Refinement process for transform coefficients of residual 8x8 blocks

Inputs to this process are

- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a variable bitDepth specifying the bit depth,
- a variable qP specifying the quantisation parameter value,
- a variable cO specifying the first coefficient index in the list of scaled transform coefficient values sTCoeff and in the list of transform coefficient values tCoeffLevel,
- a variable coeffLevel8x8 representing LumaLevel8x8, CbLevel8x8, or CrLevel8x8,
- a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,
- a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are

- a modified version of the list sTCoeff,
- a modified version of the list tCoeffLevel.

Depending on tcoeff_level_prediction_flag, the following applies.

- If tcoeff_level_prediction_flag is equal to 1, the list sTCoeff is modified by

  sTCoeff[ cO + k ] = 0  with k = 0..255  (G-144)

- Otherwise (tcoeff_level_prediction_flag is equal to 0), the list tCoeffLevel is modified by

  tCoeffLevel[ cO + k ] = 0  with k = 0..255  (G-145)

For each residual 8x8 block indexed by c8x8BlkIdx = 0..3, the following ordered steps are specified.

1. The inverse scanning process for 8x8 transform coefficients as specified in subclause 8.5.7 is invoked with coeffLevel8x8[ c8x8BlkIdx ] as the input and the outputs are transform coefficient level values as an 8x8 array c with elements $c_{ij}$. For this invocation of the process in subclause 8.5.7, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.
2. The list tCoeffLevel and the 8x8 array c are modified by

   tCoeffLevel[ cO + 64 * c8x8BlkIdx + 8 * i + j ] += $c_{ij}$  with i, j = 0..7  (G-146)

   $c_{ij}$ = tCoeffLevel[ cO + 64 * c8x8BlkIdx + 8 * i + j ]  with i, j = 0..7  (G-147)

3. The scaling process for residual 8x8 blocks as specified in subclause 8.5.12.1 is invoked with bitDepth, qP, and the 8x8 array c as the inputs and the outputs are scaled transform coefficient values as an 8x8 array d with elements $d_{ij}$.
4. The list sTCoeff is modified by

   sTCoeff[ cO + 64 * c8x8BlkIdx + 8 * i + j ] += $d_{ij}$  with i, j = 0..7  (G-148)

The bitstream shall not contain data that result in any element sTCoeff[ cO + k ] with k = 0..255 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)}-1$, inclusive.

The bitstream shall not contain data that result in any element tCoeffLevel[ cO + k ] with k = 0..255 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)}-1$, inclusive.

#### G.8.5.1.1.4 Refinement process for transform coefficients of Intra_16x16 macroblocks

This process is only invoked when base_mode_flag is equal to 0 or tcoeff_level_prediction_flag is equal to 1.

Inputs to this process are

- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a variable bitDepth specifying the bit depth,
- a variable qP specifying the quantisation parameter value,
- a variable cO specifying the first coefficient index in the list of scaled transform coefficient values sTCoeff and in the list of transform coefficient values tCoeffLevel,
- a variable coeffDCLevel representing Intra16x16DCLevel, CbIntra16x16DCLevel, or CrIntra16x16DCLevel,
- a variable coeffACLevel representing Intra16x16ACLevel, CbIntra16x16ACLevel, or CrIntra16x16ACLevel,
- a variable coeffLevel representing LumaLevel, CbLevel, or CrLevel,
- a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,
- a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are

- a modified version of the list sTCoeff,
- a modified version of the list tCoeffLevel.

NOTE 1 – When tcoeff_level_prediction_flag is equal to 0, this subclause is always invoked as part of an invocation of subclause G.8.5.1 with refinementFlag equal to 0, in which case all elements of the list tCoeffLevel are set equal to 0 before invoking this subclause.

For the DC transform coefficients of all residual 4x4 blocks, the following ordered steps are specified.

1. Depending on base_mode_flag, the 4x4 array c with elements $c_{ij}$ is derived as follows.
   - If base_mode_flag is equal to 0, the inverse scanning process for transform coefficients as specified in subclause 8.5.6 is invoked with coeffDCLevel as the input and the outputs are DC transform coefficient level values for all residual 4x4 blocks as a 4x4 array c with elements $c_{ij}$. For this invocation of the process in subclause 8.5.6, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.
   - Otherwise (base_mode_flag is equal to 1), the 4x4 array c with elements $c_{ij}$ containing DC transform coefficient level values is derived by

     $c_{ij}$ = coeffLevel[ 8 * ( i / 2 ) + 4 * ( j / 2 ) + 2 * ( i % 2 ) + ( j % 2 ) ][ 0 ] with i, j = 0..3 (G-149)

2. The list tCoeffLevel and the 4x4 array c are modified by

   tCoeffLevel[ cO + 128 * ( i / 2 ) + 64 * ( j / 2 ) + 32 * ( i % 2 ) + 16 * ( j % 2 ) ] += $c_{ij}$ with i, j = 0..3 (G-150)

   $c_{ij}$ = tCoeffLevel[ cO + 128 * ( i / 2 ) + 64 * ( j / 2 ) + 32 * ( i % 2 ) + 16 * ( j % 2 ) ] with i, j = 0..3 (G-151)

3. The scaling and transformation process for DC transform coefficients for Intra_16x16 macroblock type as specified in subclause 8.5.9 is invoked with bitDepth, qP, and c as the inputs and the output is the 4x4 array d with elements $d_{ij}$ representing scaled DC transform coefficient values for all residual 4x4 blocks.
4. The list sTCoeff is modified by

   sTCoeff[ cO + 128 * ( i / 2 ) + 64 * ( j / 2 ) + 32 * ( i % 2 ) + 16 * ( j % 2 ) ] = $d_{ij}$ with i, j = 0..3 (G-152)

For each residual 4x4 block indexed by c4x4BlkIdx = 0..15, the following ordered steps are specified.

1. Depending on base_mode_flag, the variable c4x4List, which is a list of 16 entries, is derived as follows.
   - If base_mode_flag is equal to 0, the following applies.

     c4x4List[ k ] = ( k == 0 ? 0 : coeffACLevel[ c4x4BlkIdx ][ k – 1 ] ) with k = 0..15 (G-153)

   - Otherwise (base_mode_flag is equal to 1), the following applies.

     c4x4List[ k ] = ( k == 0 ? 0 : coeffLevel[ c4x4BlkIdx ][ k ] ) with k = 0..15 (G-154)

2. The inverse scanning process for transform coefficients as specified in subclause 8.5.6 is invoked with c4x4List as the input and the outputs are transform coefficient level values as a 4x4 array e with elements $e_{ij}$. For this invocation of the process in subclause 8.5.6, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.

3. The list tCoeffLevel and the 4x4 array e are modified by

$$tCoeffLevel[\ cO + 16 * c4x4BlkIdx + 4 * i + j\ ] \mathrel{+}= e_{ij} \qquad \text{with } i, j = 0..3 \text{ and } i + j > 0 \qquad \text{(G-155)}$$

$$e_{ij} = tCoeffLevel[\ cO + 16 * c4x4BlkIdx + 4 * i + j\ ] \qquad \text{with } i, j = 0..3 \text{ and } i + j > 0 \qquad \text{(G-156)}$$

4. The scaling process for residual 4x4 blocks as specified in subclause 8.5.11.1 is invoked with bitDepth, qP, and the 4x4 array e as the inputs and the outputs are scaled transform coefficient values as a 4x4 array d with elements $d_{ij}$. During the process in subclause 8.5.11.1, the array e is treated as relating to a luma residual block coded using the Intra_16x16 prediction mode.

5. The list sTCoeff is modified by

$$sTCoeff[\ cO + 16 * c4x4BlkIdx + 4 * i + j\ ] = d_{ij} \qquad \text{with } i, j = 0..3 \text{ and } i + j > 0 \qquad \text{(G-157)}$$

NOTE 2 – The elements tCoeffLevel[ cO + 16* c4x4BlkIdx ] and sTCoeff[ cO + 16* c4x4BlkIdx ] are not modified during the process for a residual 4x4 block with index c4x4BlkIdx.

The bitstream shall not contain data that result in any element tCoeffLevel[ cO + 16* b + k ] with b = 0..15 and k = 1..15 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)}-1$, inclusive.

NOTE 3 – The elements tCoeffLevel[ cO + 16* b ] with b = 0..15 can exceed the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)}-1$, inclusive.

#### G.8.5.1.2 Refinement process for chroma transform coefficients

Inputs to this process are

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable cTrafo specifying the transform type,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,

– a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are

– a modified version of the list sTCoeff,

– a modified version of the list tCoeffLevel.

For both chroma components indexed by iCbCr = 0..1, the following applies.

– If ChromaArrayType is equal to 1 or 2, the following applies.

  – If cTrafo is equal to T_PCM, the assignment process for chroma transform coefficient values for I_PCM macroblocks as specified in subclause G.8.5.1.2.1 is invoked with iCbCr, sTCoeff, and tCoeffLevel as the inputs and the outputs are modified versions of sTCoeff and tCoeffLevel.

  – Otherwise (cTrafo is not equal to T_PCM), the refinement process for chroma transform coefficients with ChromaArrayType equal to 1 or 2 as specified in subclause G.8.5.1.2.2 is invoked with iCbCr, fieldMbFlag, sTCoeff, and tCoeffLevel as the inputs and the outputs are modified versions of the lists sTCoeff and tCoeffLevel.

– Otherwise (ChromaArrayType is equal to 3), the refinement process for luma transform coefficients or chroma transform coefficients with ChromaArrayType equal to 3 as specified in subclause G.8.5.1.1 is invoked with iYCbCr set equal to (1 + iCbCr), fieldMbFlag, cTrafo, sTCoeff, and tCoeffLevel as the inputs and the outputs are modified versions of the lists sTCoeff and tCoeffLevel.

##### G.8.5.1.2.1 Assignment process for chroma transform coefficient values for I_PCM macroblocks

Inputs to this process are

– a variable iCbCr specifying the chroma component,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,

– a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are

– a modified version of the list sTCoeff,

– a modified version of the list tCoeffLevel.

The variable numC is set equal to (MbWidthC * MbHeightC) and the variable cCO is set equal to (iCbCr * numC).

When base_mode_flag is equal to 0, the lists tCoeffLevel and sTCoeff are modified by

$$tCoeffLevel[\ 256 + cCO + k\ ] = 0 \qquad \text{with } k = 0..(numC - 1) \tag{G-158}$$

$$sTCoeff[\ 256 + cCO + k\ ] = pcm\_sample\_chroma[\ cCO + k\ ] \qquad \text{with } k = 0..(numC - 1) \tag{G-159}$$

#### G.8.5.1.2.2 Refinement process for chroma transform coefficients with ChromaArrayType equal to 1 or 2

This process is only invoked when ChromaArrayType is equal to 1 or 2.

Inputs to this process are

– a variable iCbCr specifying the chroma component,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,

– a list of transform coefficient level values tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are

– a modified version of the list sTCoeff,

– a modified version of the list tCoeffLevel.

The variables nW, nH, numB, cO, and qP are derived by

$$nW = MbWidthC / 4 \tag{G-160}$$
$$nH = MbHeightC / 4 \tag{G-161}$$
$$numB = nW * nH \tag{G-162}$$
$$cO = 256 + ( iCbCr * MbWidthC * MbHeightC ) \tag{G-163}$$
$$qP = ( iCbCr == 0\ ?\ QP'_{Cb}\ :\ QP'_{Cr} ) \tag{G-164}$$

Depending on tcoeff_level_prediction_flag, the following applies.

– If tcoeff_level_prediction_flag is equal to 1, the list sTCoeff is modified by

$$sTCoeff[\ cO + k\ ] = 0 \qquad \text{with } k = 0..(MbWidthC * MbHeightC - 1) \tag{G-165}$$

– Otherwise (tcoeff_level_prediction_flag is equal to 0), the list tCoeffLevel is modified by

$$tCoeffLevel[\ cO + k\ ] = 0 \qquad \text{with } k = 0..(MbWidthC * MbHeightC - 1) \tag{G-166}$$

For the chroma DC transform coefficients of all residual 4x4 chroma blocks, the following ordered steps are specified.

1. Depending ChromaArrayType, the (nW)x(nH) array c with elements $c_{ij}$ is derived as follows.

   – If ChromaArrayType is equal to 1,

   $$c_{ij} = ChromaDCLevel[\ iCbCr\ ][\ 2 * i + j\ ] \qquad \text{with } i = 0..(nH - 1),\ j = 0..(nW - 1) \tag{G-167}$$

   – Otherwise (ChromaArrayType is equal to 2),

   $$c_{ij} = ChromaDCLevel[\ iCbCr\ ][\ scan422ChromaDC[\ 2 * i + j\ ]\ ] \qquad \text{with } i = 0..(nH - 1),\ j = 0..(nW - 1),\ \text{and } scan422ChromaDC = \{\ 0, 2, 1, 5, 3, 6, 4, 7\ \} \tag{G-168}$$

2. The list tCoeffLevel and the (nW)x(nH) array c are modified by

   $$tCoeffLevel[\ cO + 32 * i + 16 * j\ ] \mathrel{+}= c_{ij} \qquad \text{with } i = 0..(nH - 1),\ j = 0..(nW - 1) \tag{G-169}$$

   $$c_{ij} = tCoeffLevel[\ cO + 32 * i + 16 * j\ ] \qquad \text{with } i = 0..(nH - 1),\ j = 0..(nW - 1) \tag{G-170}$$

3. The variable $qP_{DC}$ is derived by

$$qP_{DC} = ( \text{ChromaArrayType} == 1 \ ? \ qP \ : \ qP + 3 ) \quad \text{(G-171)}$$

4. The (nW)x(nH) array d with elements $d_{ij}$ representing scaled chroma DC transform coefficient values for all residual 4x4 chroma blocks is derived by

$$d_{ij} = c_{ij} * ( \text{LevelScale}( qP_{DC} \% 6, 0, 0 ) << ( qP_{DC} / 6 ) ) \quad \text{with } i = 0..(nH - 1),\ j = 0..(nW - 1) \quad \text{(G-172)}$$

5. The list sTCoeff is modified by

$$sTCoeff[ cO + 32 * i + 16 * j ] \mathrel{+}= d_{ij} \quad \text{with } i = 0..(nH - 1),\ j = 0..(nW - 1) \quad \text{(G-173)}$$

For each residual 4x4 chroma block indexed by c4x4BlkIdx = 0..numB – 1, the following ordered steps are specified.

1. The variable c4x4List, which is a list of 16 entries, is derived by

$$c4x4List[ k ] = ( k == 0 \ ? \ 0 \ : \ ChromaACLevel[ iCbCr ][ c4x4BlkIdx ][ k - 1 ] ) \quad \text{with } k = 0..15 \quad \text{(G-174)}$$

2. The inverse scanning process for transform coefficients as specified in subclause 8.5.6 is invoked with c4x4List as the input and the outputs are chroma transform coefficient level values as a 4x4 array e with elements $e_{ij}$. During the process in subclause 8.5.6, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.

3. The list tCoeffLevel and the 4x4 array e are modified by

$$tCoeffLevel[ cO + 16 * c4x4BlkIdx + 4 * i + j ] \mathrel{+}= e_{ij} \quad \text{with } i, j = 0..3 \text{ and } i + j > 0 \quad \text{(G-175)}$$

$$e_{ij} = tCoeffLevel[ cO + 16 * c4x4BlkIdx + 4 * i + j ] \quad \text{with } i, j = 0..3 \text{ and } i + j > 0 \quad \text{(G-176)}$$

4. The scaling process for residual 4x4 blocks as specified in subclause 8.5.11.1 is invoked with $BitDepth_C$, qP, and the 4x4 array e as the inputs and the outputs are scaled chroma transform coefficient values as a 4x4 array d of with elements $d_{ij}$. During the process in subclause 8.5.11.1, the array e is treated as relating to a chroma residual block.

5. The list sTCoeff is modified by

$$sTCoeff[ cO + 16 * c4x4BlkIdx + 4 * i + j ] \mathrel{+}= d_{ij} \quad \text{with } i, j = 0..3 \text{ and } i + j > 0 \quad \text{(G-177)}$$

NOTE 1 – The elements tCoeffLevel[ cO + 16* c4x4BlkIdx ] and sTCoeff[ cO + 16* c4x4BlkIdx ] are not modified during the process for a residual 4x4 chroma block with index c4x4BlkIdx.

The bitstream shall not contain data that result in any element sTCoeff[ cO + 16* b + k ] with b = 0..(numB – 1) and k = 1..15 that exceeds the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)}-1$, inclusive.

The bitstream shall not contain data that result in any element tCoeffLevel[ cO + 16* b + k ] with b = 0..(numB – 1) and k = 1..15 that exceeds the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)}-1$, inclusive.

NOTE 2 – The elements tCoeffLevel[ cO + 16* b ] and sTCoeff[ cO + 16* b ] with b = 0..(numB – 1) can exceed the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)}-1$, inclusive.

### G.8.5.2 Transform coefficient level scaling process prior to transform coefficient refinement

Inputs to this process are

- a variable cTrafo specifying the luma transform type for the current macroblock,
- a list tCoeffLevel with (256 + 2 * MbWidthC * MbHeightC) elements specifying transform coefficient level values for the current macroblock,
- a variable $tQP_Y$ specifying the luma quantisation parameter for the current macroblock,
- a variable $refQP_Y$ specifying the quantisation parameter for the macroblock of the reference layer representation,
- when ChromaArrayType is not equal to 0, two variables $tQP_{Cb}$ and $tQP_{Cr}$ specifying chroma quantisation parameters for the current macroblock,
- when ChromaArrayType is not equal to 0, two variables $refQP_{Cb}$ and $refQP_{Cr}$ specifying chroma quantisation parameters for the macroblock of the reference layer representation.

Output of this process is a modified version of the list tCoeffLevel.

Table G-5 specifies the scale values cS for transform coefficient level scaling.

**Table G-5 – Scale values cS for transform coefficient level scaling**

| ( refQP – cQP + 54 ) % 6 | scale value cS |
|---|---|
| 0 | 8 |
| 1 | 9 |
| 2 | 10 |
| 3 | 11 |
| 4 | 13 |
| 5 | 14 |

The variable iYCbCr proceeds over the values from 0 to (ChromaArrayType == 0 ? 0 : 2), inclusive, and for each value of iYCbCr, the following ordered steps are specified.

1. The variables cO, iMax, cQP, and refQP are derived by

$$cO = ( iYCbCr == 0 ? 0 : 256 + ( iYCbCr - 1 ) * MbWidthC * MbHeightC ) \quad (G\text{-}178)$$
$$iMax = ( iYCbCr == 0 ? 255 : MbWidthC * MbHeightC - 1 ) \quad (G\text{-}179)$$
$$cQP = ( iYCbCr == 0 ? tQP_Y : ( iYCbCr == 1 ? tQP_{Cb} : tQP_{Cr} ) ) \quad (G\text{-}180)$$
$$refQP = ( iYCbCr == 0 ? refQP_Y : ( iYCbCr == 1 ? refQP_{Cb} : refQP_{Cr} ) ) \quad (G\text{-}181)$$

2. The variable cS is set as specified in Table G-5 using the values of refQP and cQP.

3. The variable rShift is calculated by

$$rShift = ( refQP - cQP + 54 ) / 6 \quad (G\text{-}182)$$

4. The list tCoeffLevel of transform coefficient level values is modified by

$$tCoeffLevel[ cO + i ] = ( ( cS * tCoeffLevel[ cO + i ] ) << rShift ) >> 12 \quad \text{with } i = 0..iMax \quad (G\text{-}183)$$

The following constraints shall be obeyed.

a) When cTrafo is equal to T_4x4 or T_8x8, the bitstream shall not contain data that result in any element tCoeffLevel[ k ] with k = 0..255 that exceeds the range of integer values from $-2^{(7 + BitDepth_Y)}$ to $2^{(7 + BitDepth_Y)}-1$, inclusive.

b) When cTrafo is equal to T_16x16, the bitstream shall not contain data that result in any element tCoeffLevel[ 16* b + k ] with b = 0..15 and k = 1..15 that exceeds the range of integer values from $-2^{(7 + BitDepth_Y)}$ to $2^{(7 + BitDepth_Y)}-1$, inclusive.

NOTE 1 – When cTrafo is equal to T_16x16, the elements tCoeffLevel[ 16* b ] with b = 0..15 can exceed the range of integer values from $-2^{(7 + BitDepth_Y)}$ to $2^{(7 + BitDepth_Y)}-1$, inclusive.

c) When ChromaArrayType is equal to 1 or 2 and cTrafo is not equal to T_PCM, the bitstream shall not contain data that result in any element tCoeffLevel[ 256 + 16* b + k ] with b = 0..(MbWidthC * MbHeightC / 8 – 1), and k = 1..15 that exceeds the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)}-1$, inclusive.

NOTE 2 When ChromaArrayType is equal to 1 or 2 and cTrafo is not equal to T_IPCM, the elements tCoeffLevel[ 256 + 16* b ] with b = 0..(MbWidthC * MbHeightC / 8 – 1) can exceed the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)}-1$, inclusive.

d) When ChromaArrayType is equal to 3 and cTrafo is equal to T_4x4 or T_8x8, the bitstream shall not contain data that result in any element tCoeffLevel[ 256 + k ] with k = 0..511 that exceeds the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)}-1$, inclusive.

e) When ChromaArrayType is equal to 3 and cTrafo is T_16x16, the bitstream shall not contain data that result in any element tCoeffLevel[ 256 + 16* b + k ] with b = 0..31 and k = 1..15 that exceeds the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)}-1$, inclusive.

NOTE 3 – When ChromaArrayType is equal to 3 and cTrafo is equal to T_16x16, the elements tCoeffLevel[ 256 + 16* b ] with b = 0..(MbWidthC * MbHeightC / 8 – 1) can exceed the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)}-1$, inclusive.

NOTE 4 – When $tQP_Y$ is less than 10 and cTrafo is equal to T_16x16, the range of values that can be represented by an alternative representation of the bitstream with entropy_coding_mode_flag equal to 0 and profile_idc equal to 66, 77, or 88, may not be sufficient to represent the full range of values of the elements tCoeffLevel[ 16 * b ] with b = 0..15 that could be necessary to form a close approximation of the content of any possible source picture.

NOTE 5 – When ChromaArrayType is equal to 1 or 2 and $tQP_{CX}$ with CX being replaced by Cb and Cr is less than 4, the range of values that can be represented by an alternative representation of the bitstream with entropy_coding_mode_flag equal to 0 and profile_idc equal to 66, 77, or 88, may not be sufficient to represent the full range of values of the elements tCoeffLevel[ 256 + 16 * b ] with b = 0..(MbWidthC * MbHeightC / 8 – 1) that could be necessary to form a close approximation of the content of any possible source picture.

### G.8.5.3 Residual construction and accumulation process

Inputs to this process are

- a variable accumulationFlag specifying whether the constructed residual sample values for the current macroblock are combined with the existent residual sample value for the macroblock,
- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a variable cTrafo specifying the transform type,
- a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements,
- a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array $picRes_L$ containing residual luma sample values for the current layer representation,
- when ChromaArrayType is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picRes_{Cb}$ and $picRes_{Cr}$ containing residual chroma sample values for the current layer representation.

Outputs of this process are

- a modified version of the array $picRes_L$,
- when ChromaArrayType is not equal to 0, modified versions of the arrays $picRes_{Cb}$ and $picRes_{Cr}$.

The construction process for luma residuals as specified in subclause G.8.5.3.1 is invoked with cTrafo and sTCoeff as the inputs and the outputs are residual luma sample values as a 16x16 array $mbRes_L$.

When ChromaArrayType is not equal to 0, the construction process for chroma residuals as specified in subclause G.8.5.3.2 is invoked with cTrafo and sTCoeff as the inputs and the outputs are residual chroma sample values as two (MbWidthC)x(MbHeightC) arrays $mbRes_{Cb}$ and $mbRes_{Cr}$.

When accumulationFlag is equal to 1, the following ordered steps are specified.

1. The macroblock sample array extraction process as specified in subclause G.8.5.4.2 is invoked with fieldMbFlag, $picRes_L$, and, when ChromaArrayType is equal to 0, $picRes_{Cb}$ and $picRes_{Cr}$ as the inputs and the outputs are a 16x16 array $refLayerMbRes_L$ and, when ChromaArrayType is not equal to 0, two (MbWidthC)x(MbHeightC) arrays $refLayerMbRes_{Cb}$ and $refLayerMbRes_{Cr}$.
2. All elements $mbRes_L[ x, y ]$ of the 16x16 array $mbRes_L$ with x, y = 0..15 are modified by

$$mbRes_L[ x, y ] = Clip3( yMin, yMax, mbRes_L[ x, y ] + refLayerMbRes_L[ x, y ] ) \quad \text{(G-184)}$$

with

$$yMin = -( 1 << BitDepth_Y ) + 1 \quad \text{(G-185)}$$
$$yMax = ( 1 << BitDepth_Y ) - 1 \quad \text{(G-186)}$$

3. When ChromaArrayType is not equal to 0, for CX being replaced by Cb and Cr, all elements $mbRes_{CX}[ x, y ]$ of the (MbWidthC)x(MbHeightC) array $mbRes_{CX}$ with x = 0..(MbWidthC – 1) and y = 0..(MbHeightC – 1) are modified by

$$mbRes_{CX}[ x, y ] = Clip3( cMin, cMax, mbRes_{CX}[ x, y ] + refLayerMbRes_{CX}[ x, y ] ) \quad \text{(G-187)}$$

with

$$cMin = -( 1 << BitDepth_C ) + 1 \quad \text{(G-188)}$$
$$cMax = ( 1 << BitDepth_C ) - 1 \quad \text{(G-189)}$$

The picture sample array construction process as specified in subclause G.8.5.4.1 is invoked with fieldMbFlag, $mbRes_L$, $picRes_L$, and, when ChromaArrayType is not equal to 0, $mbRes_{Cb}$, $mbRes_{Cr}$, $picRes_{Cb}$, and $picRes_{Cr}$ as the inputs and the

outputs are a modified version of the array $picRes_L$ and, when ChromaArrayType is not equal to 0, modified versions of the arrays $picRes_{Cb}$ and $picRes_{Cr}$.

#### G.8.5.3.1 Construction process for luma residuals or chroma residuals with ChromaArrayType equal to 3

Inputs to this process are

- a variable iYCbCr specifying the colour component (when present),
- a variable cTrafo specifying the transform type,
- a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are residual sample values as a 16x16 array mbRes with elements mbRes[ x, y ].

When iYCbCr is not present as input to this subclause, it is inferred to be equal to 0.

Depending on iYCbCr, the variables bitDepth and cO are derived as follows.

- If iYCbCr is equal to 0, bitDepth is set equal to $BitDepth_Y$ and cO is set equal to 0.
- Otherwise, if iYCbCr is equal to 1, bitDepth is set equal to $BitDepth_C$ and cO is set equal to 256.
- Otherwise (iYCbCr is equal to 2), bitDepth is set equal to $BitDepth_C$ and cO is set equal to (256 + MbWidthC * MbHeightC).

Depending on cTrafo, the 16x16 array mbRes is derived as follows.

- If cTrafo is equal to T_PCM, the construction process for luma residuals or chroma residuals with ChromaArrayType equal to 3 of I_PCM macroblocks as specified in subclause G.8.5.3.1.1 is invoked with cO and sTCoeff as the inputs and the output is the 16x16 array mbRes of residual sample values.
- Otherwise, if cTrafo is equal to T_4x4, the construction process for residual 4x4 blocks as specified in subclause G.8.5.3.1.2 is invoked with bitDepth, cO, and sTCoeff as the inputs and the output is the 16x16 array mbRes of residual sample values.
- Otherwise, if cTrafo is equal to T_8x8, the construction process for residual 8x8 blocks as specified in subclause G.8.5.3.1.3 is invoked with bitDepth, cO, and sTCoeff as the inputs and the output is the 16x16 array mbRes of residual sample values.
- Otherwise (cTrafo is equal to T_16x16), the construction process for residuals of Intra_16x16 macroblocks as specified in subclause G.8.5.3.1.4 is invoked with bitDepth, cO, and sTCoeff as the inputs and the output is the 16x16 array mbRes of residual sample values.

##### G.8.5.3.1.1 Construction process for luma residuals or chroma residuals with ChromaArrayType equal to 3 of I_PCM macroblocks

Inputs to this process are

- a variable cO specifying the first coefficient index in the list of scaled transform coefficient values sTCoeff,
- a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are residual sample values as a 16x16 array mbRes with elements mbRes[ x, y ].

The 16x16 array mbRes is derived by

$$mbRes[ x, y ] = sTCoeff[ cO + y * 16 + x ] \quad \text{with } x, y = 0..15 \tag{G-190}$$

##### G.8.5.3.1.2 Construction process for residual 4x4 blocks

Inputs to this process are

- a variable bitDepth specifying the bit depth,
- a variable cO specifying the first coefficient index in the list of scaled transform coefficient values sTCoeff,
- a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are residual sample values as a 16x16 array mbRes with elements mbRes[ x, y ].

For each residual 4x4 block indexed by c4x4BlkIdx = 0..15, the following ordered steps are specified.

1. The 4x4 array d with elements $d_{ij}$ is derived by

$$d_{ij} = sTCoeff[\ cO + 16 * c4x4BlkIdx + 4 * i + j\ ] \qquad \text{with } i, j = 0..3 \tag{G-191}$$

2. The transformation process for residual 4x4 blocks as specified in subclause 8.5.11.2 is invoked with bitDepth and the 4x4 array d as the inputs and the outputs are residual sample value as a 4x4 array r with elements $r_{ij}$.
3. The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with c4x4BlkIdx as the input and the output is assigned to ( xP, yP ).
4. The elements mbRes[ x, y ] of the 16x16 array mbRes with x = xP..(xP + 3) and y = yP..(yP + 3) are derived by

$$mbRes[\ xP + j, yP + i\ ] = r_{ij} \qquad \text{with } i, j = 0..3 \tag{G-192}$$

#### G.8.5.3.1.3 Construction process for residual 8x8 blocks

Inputs to this process are

– a variable bitDepth specifying the bit depth,

– a variable cO specifying the first coefficient index in the list of scaled transform coefficient values sTCoeff,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are residual sample values as a 16x16 array mbRes with elements mbRes[ x, y ].

For each residual 8x8 block indexed by c8x8BlkIdx = 0..3, the following ordered steps are specified.

1. The 8x8 array d with elements $d_{ij}$ is derived by

$$d_{ij} = sTCoeff[\ cO + 64 * c8x8BlkIdx + 8 * i + j\ ] \qquad \text{with } i, j = 0..7 \tag{G-193}$$

2. The transformation process for residual 8x8 blocks as specified in subclause 8.5.12.2 is invoked with bitDepth and the 8x8 array d as the inputs and the outputs are residual sample values as an 8x8 array r with elements $r_{ij}$.
3. The inverse 8x8 luma block scanning process as specified in subclause 6.4.5 is invoked with c8x8BlkIdx as the input and the output is assigned to ( xP, yP ).
4. The elements mbRes[ x, y ] of the 16x16 array mbRes with x = xP..(xP + 7) and y = yP..(yP + 7) are derived by

$$mbRes[\ xP + j, yP + i\ ] = r_{ij} \qquad \text{with } i, j = 0..7 \tag{G-194}$$

#### G.8.5.3.1.4 Construction process for residuals of Intra_16x16 macroblocks

Inputs to this process are

– a variable bitDepth specifying the bit depth,

– a variable cO specifying the first coefficient index in the list of scaled transform coefficient values sTCoeff,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are residual sample values as a 16x16 array mbRes with elements mbRes[ x, y ].

For each residual 4x4 block indexed by c4x4BlkIdx = 0..15, the following ordered steps are specified.

1. The 4x4 array d with elements $d_{ij}$ is derived by.

$$d_{ij} = sTCoeff[\ cO + 16 * c4x4BlkIdx + 4 * i + j\ ] \qquad \text{with } i, j = 0..3 \tag{G-195}$$

2. The transformation process for residual 4x4 blocks as specified in subclause 8.5.11.2 is invoked with bitDepth and the 4x4 array d as the inputs and the outputs are residual sample values as a 4x4 array r with elements $r_{ij}$.
3. The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with c4x4BlkIdx as the input and the output is assigned to ( xP, yP ).
4. The elements mbRes[ x, y ] of the 16x16 array mbRes with x = xP..(xP + 3) and y = yP..(yP + 3) are derived by

$$mbRes[\ xP + j, yP + i\ ] = r_{ij} \qquad \text{with } i, j = 0..3 \tag{G-196}$$

#### G.8.5.3.2 Construction process for chroma residuals

Inputs to this process are

– a variable cTrafo specifying the transform type,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Outputs of this process are residual chroma sample values as two (MbWidthC)x(MbHeightC) arrays $mbRes_{Cb}$ and $mbRes_{Cr}$ with elements $mbRes_{Cb}$[ x, y ] and $mbRes_{Cr}$[ x, y ], respectively.

For both chroma components indexed by iCbCr = 0..1 and for CX being replaced by Cb for iCbCr equal to 0 and Cr for iCbCr equal to 1, the following applies.

– If ChromaArrayType is equal to 1 or 2, the following applies.

  – If cTrafo is equal to T_PCM, the construction process for chroma residuals of I_PCM macroblocks as specified in subclause G.8.5.3.2.1 is invoked with iCbCr and sTCoeff as the inputs and the output is the (MbWidthC)x(MbHeightC) array $mbRes_{CX}$ of residual chroma sample values.

  – Otherwise (cTrafo is not equal to T_PCM), the construction process for chroma residuals with ChromaArrayType equal to 1 or 2 as specified in subclause G.8.5.3.2.2 is invoked with iCbCr and sTCoeff as the inputs and the output is the (MbWidthC)x(MbHeightC) array $mbRes_{CX}$ of residual chroma sample values.

– Otherwise (ChromaArrayType is equal to 3), the construction process for luma residuals or chroma residuals with ChromaArrayType equal to 3 as specified in subclause G.8.5.3.1 is invoked with iYCbCr set equal to (1 + iCbCr), cTrafo, and sTCoeff as the inputs and the output is the (MbWidthC)x(MbHeightC) array $mbRes_{CX}$ of residual chroma sample values.

##### G.8.5.3.2.1 Construction process for chroma residuals of I_PCM macroblocks

Inputs to this process are

– a variable iCbCr specifying the chroma component,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Output of this process are residual chroma sample values as a (MbWidthC)x(MbHeightC) array mbRes with elements mbRes[ x, y ].

The (MbWidthC)x(MbHeightC) array mbRes is derived by

$$mbRes[ x, y ] = sTCoeff[ 256 + iCbCr * MbWidth * MbHeight + y * MbWidthC + x ] \quad \text{with } x = 0..(MbWidthC - 1) \text{ and } y = 0..(MbHeightC - 1) \tag{G-197}$$

##### G.8.5.3.2.2 Construction process for chroma residuals with ChromaArrayType equal to 1 or 2

This process is only invoked when ChromaArrayType is equal to 1 or 2.

Inputs to this process are

– a variable iCbCr specifying the chroma component,

– a list of scaled transform coefficient values sTCoeff with (256 + 2 * MbWidthC * MbHeightC) elements.

Output of this process are residual chroma sample values as a (MbWidthC)x(MbHeightC) array mbRes with elements mbRes[ x, y ].

The variables nW, nH, numB, and cO are derived by

$$nW = MbWidthC / 4 \tag{G-198}$$
$$nH = MbHeightC / 4 \tag{G-199}$$
$$numB = nW * nH \tag{G-200}$$
$$cO = 256 + ( iCbCr * MbWidthC * MbHeightC ) \tag{G-201}$$

For the chroma DC transform coefficients of all residual 4x4 chroma blocks, the following ordered steps are specified.

1. The (nW)x(nH) array c with the elements $c_{ij}$ is derived by

$$c_{ij} = sTCoeff[ cO + 32 * i + 16 * j ] \quad \text{with } i = 0..(nH - 1),\ j = 0..(nW - 1) \tag{G-202}$$

2. The transformation process for chroma DC transform coefficients as specified in subclause 8.5.10.1 is invoked with $BitDepth_C$ and the (nW)x(nH) array c as the inputs and the outputs are DC values for all residual 4x4 chroma blocks as a (nW)x(nH) array f with elements $f_{ij}$.

3. Depending on ChromaArrayType, the (nW)x(nH) array dcC with elements $dcC_{ij}$ is derived as follows.

   – If ChromaArrayType is equal to 1,

   $$dcC_{ij} = f_{ij} >> 5 \qquad \text{with } i = 0..(nH - 1),\ j = 0..(nW - 1) \tag{G-203}$$

   – Otherwise (ChromaArrayType is equal to 2),

   $$dcC_{ij} = ( f_{ij} + ( 1 << 5 ) ) >> 6 \qquad \text{with } i = 0..(nH - 1),\ j = 0..(nW - 1) \tag{G-204}$$

The bitstream shall not contain data that result in any element $dcC_{ij}$ of dcC with $i = 0..(nH - 1)$ and $j = 0..(nW - 1)$ that exceeds the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)}-1$, inclusive.

NOTE – For the layer representation with dependency_id equal to 0 and quality_id equal to 0, successive invocations of subclause G.8.5.1.2 (as part of an invocation of subclause G.8.5.1) and this subclause yield an array dcC that is identical to the array dcC that would be obtained by an invocation of subclause 8.5.10. However, the intermediate values $c_{ij}$ and $f_{ij}$ with $i = 0..(nH - 1)$ and $j = 0..(nW - 1)$ that are derived in this subclause can exceed the range of integer values from $-2^{(7 + BitDepth_C)}$ to $2^{(7 + BitDepth_C)}-1$, inclusive.

For each residual 4x4 chroma block indexed by $c4x4BlkIdx = 0..(numB - 1)$, the following ordered steps are specified.

1. The 4x4 array d with elements $d_{ij}$ is derived as follows.

   – The element $d_{00}$ is derived by

   $$d_{00} = dcY_{kl} \quad \text{with } k = c4x4BlkIdx / 2 \text{ and } l = c4x4BlkIdx \% 2 \tag{G-205}$$

   – The elements $d_{ij}$ with $i, j = 0..3$ and $i + j > 0$ are derived by

   $$d_{ij} = sTCoeff[\ cO + 16 * c4x4BlkIdx + 4 * i + j\ ] \tag{G-206}$$

2. The transformation process for residual 4x4 blocks as specified in subclause 8.5.11.2 is invoked with $BitDepth_C$ and the 4x4 array d as the inputs and the outputs are residual chroma sample values as a 4x4 array r with elements $r_{ij}$.

3. The chroma location ( xP, yP ) is derived by

   $$xP = 4 * ( c4x4BlkIdx \% 2 ) \tag{G-207}$$
   $$yP = 4 * ( c4x4BlkIdx / 2 ) \tag{G-208}$$

4. The elements mbRes[ x, y ] of the (MbWidthC)x(MbHeightC) array mbRes with $x = xP..(xP + 3)$ and $y = yP..(yP + 3)$ are derived by

   $$mbRes[\ xP + j, yP + i\ ] = r_{ij} \qquad \text{with } i, j = 0..3 \tag{G-209}$$

### G.8.5.4 Sample array accumulation process

Inputs to this process are

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a $(PicWidthInSamples_L)$x$(PicHeightInSamples_L)$ array $picRes_L$ containing residual luma sample values for the current layer representation,

– a $(PicWidthInSamples_L)$x$(PicHeightInSamples_L)$ array $picSamples_L$ containing constructed luma sample values for the current layer representation,

– when ChromaArrayType is not equal to 0, two $(PicWidthInSamples_C)$x$(PicHeightInSamples_C)$ arrays $picRes_{Cb}$ and $picRes_{Cr}$ containing residual chroma sample values for the current layer representation,

– when ChromaArrayType is not equal to 0, two $(PicWidthInSamples_C)$x$(PicHeightInSamples_C)$ arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ containing constructed chroma sample values for the current layer representation.

Outputs of this process are

– a modified version of the array $picSamples_L$,

– when ChromaArrayType is not equal to 0, modified versions of the arrays $picSamples_{Cb}$ and $picSamples_{Cr}$.

The macroblock sample array extraction process as specified in subclause G.8.5.4.2 is invoked with fieldMbFlag, $picRes_L$, and, when ChromaArrayType is not equal to 0, $picRes_{Cb}$ and $picRes_{Cr}$ as the inputs and the outputs are assigned to $mbRes_L$ and, when ChromaArrayType is not equal to 0, $mbRes_{Cb}$ and $mbRes_{Cr}$.

The macroblock sample array extraction process as specified in subclause G.8.5.4.2 is invoked with fieldMbFlag, $picSamples_L$, and, when ChromaArrayType is not equal to 0, $picSamples_{Cb}$ and $picSamples_{Cr}$ as the inputs and the outputs are assigned to $mbPred_L$ and, when ChromaArrayType is not equal to 0, $mbPred_{Cb}$ and $mbPred_{Cr}$.

The 16x16 array $mbSamples_L$ is derived by

$$mbSamples_L[ x, y ] = Clip1_Y( mbPred_L[ x, y ] + mbRes_L[ x, y ] ) \quad \text{with } x, y = 0..15 \quad \text{(G-210)}$$

When ChromaArrayType is not equal to 0, for CX being replaced by Cb and Cr, the (MbWidthC)x(MbHeightC) array $mbSamples_{CX}$ is derived by

$$mbSamples_{CX}[ x, y ] = Clip1_C( mbPred_{CX}[ x, y ] + mbRes_{CX}[ x, y ] ) \quad \text{with } x = 0..(MbWidthC - 1) \text{ and } y = 0..(MbHeightC - 1) \quad \text{(G-211)}$$

The picture sample array construction process as specified in subclause G.8.5.4.1 is invoked with fieldMbFlag, $mbSamples_L$, $picSamples_L$, and, when ChromaArrayType is not equal to 0, $mbSamples_{Cb}$, $mbSamples_{Cr}$, $picSamples_{Cb}$, and $picSamples_{Cr}$ as inputs and the outputs are a modified version of $picSamples_L$ and, when ChromaArrayType is not equal to 0, modified versions of $picSamples_{Cb}$ and $picSamples_{Cr}$.

**G.8.5.4.1 Picture sample array construction process**

Inputs to this process are

- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a 16x16 array $mbArray_L$ containing luma sample values for the current macroblock,
- a ($PicWidthInSamples_L$)x($PicWidthInHeight_L$) array $picArray_L$ containing luma sample values for the current layer representation,
- when ChromaArrayType is not equal to 0, two (MbWidthC)x(MbHeightC) arrays $mbArray_{Cb}$ and $mbArray_{Cr}$ containing chroma sample values for the current macroblock,
- when ChromaArrayType is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picArray_{Cb}$ and $picArray_{Cr}$ containing chroma sample values for the current layer representation.

Outputs of this process are

- a modified version of the array $picArray_L$,
- when ChromaArrayType is not equal to 0, modified versions of the arrays $picArray_{Cb}$ and $picArray_{Cr}$.

The picture sample array construction process for a colour component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag, mbW set equal to 16, mbH set equal to 16, $mbArray_L$, and $picArray_L$ as the inputs and the output is a modified version of the array $picArray_L$.

When ChromaArrayType is not equal to 0, for CX being replaced with Cr and Cb, the picture sample array construction process for a colour component as specified in subclause G.8.5.4.3 is invoked with fieldMbFlag, mbW set equal to MbWidthC, mbH set equal to MbHeightC, $mbArray_{CX}$, and $picArray_{CX}$ as the inputs and the output is a modified version of the array $picArray_{CX}$.

**G.8.5.4.2 Macroblock sample array extraction process**

Inputs to this process are

- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a ($PicWidthInSamples_L$)x($PicWidthInHeight_L$) array $picArray_L$ containing luma sample values for the current layer representation
- when ChromaArrayType is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picArray_{Cb}$ and $picArray_{Cr}$ containing chroma sample values for the current layer representation.

Outputs of this process are

- a 16x16 array $mbArray_L$ containing luma sample values for the current macroblock,

– when ChromaArrayType is not equal to 0, two (MbWidthC)x(MbHeightC) arrays $mbArray_{Cb}$ and $mbArray_{Cr}$ containing chroma sample values for the current macroblock.

The macroblock sample array extraction process for a colour component as specified in subclause G.8.5.4.4 is invoked with fieldMbFlag, mbW set equal to 16, mbH set equal to 16, and $picArray_L$ as the inputs and the output is assigned to $mbArray_L$.

When ChromaArrayType is not equal to 0, for CX being replaced with Cr and Cb, the macroblock sample array extraction process for a colour component as specified in subclause G.8.5.4.4 is invoked with fieldMbFlag, mbW set equal to MbWidthC, mbH set equal to MbHeightC, and $picArray_{CX}$ as the inputs and the output is assigned to $mbArray_{CX}$.

**G.8.5.4.3 Picture sample array construction process for a colour component**

Inputs to this process are

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable mbW specifying the width of a macroblock colour component in samples,

– a variable mbH specifying the height of a macroblock colour component in samples,

– an (mbW)x(mbH) array mbArray containing sample values for a colour component of the current macroblock,

– an (mbW * PicWidthInMbs)x(mbH * PicHeightInMbs) array picArray containing sample values for a colour component of the current layer representation.

Output of this process is a modified version of the array picArray.

The inverse macroblock scanning process as specified in subclause 6.4.1 is invoked with CurrMbAddr as the input and the output is assigned to ( xO, yO ). During the process in subclause 6.4.1, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.

The sample location ( xP, yP ) is derived by

$$xP = ( xO >> 4 ) * mbW \quad \text{(G-212)}$$
$$yP = ( ( yO >> 4 ) * mbH ) + ( yO \% 2 ) \quad \text{(G-213)}$$

Depending on the variables MbaffFrameFlag and fieldMbFlag, the array picArray is modified as follows.

– If MbaffFrameFlag is equal to 1 and fieldMbFlag is equal to 1,

$$picArray[ xP + x, yP + 2 * y ] = mbArray[ x, y ] \quad \text{with } x = 0..(mbW - 1),\ y = 0..(mbH - 1) \quad \text{(G-214)}$$

– Otherwise (MbaffFrameFlag is equal to 0 or fieldMbFlag is equal to 0),

$$picArray[ xP + x, yP + y ] = mbArray[ x, y ] \quad \text{with } x = 0..(mbW - 1),\ y = 0..(mbH - 1) \quad \text{(G-215)}$$

**G.8.5.4.4 Macroblock sample array extraction process for a colour component**

Inputs to this process are

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable mbW specifying the width of a macroblock colour component in samples,

– a variable mbH specifying the height of a macroblock colour component in samples,

– an (mbW * PicWidthInMbs)x(mbH * PicHeightInMbs) array picArray containing sample values for a colour component of the current layer representation.

Output of this process is an (mbW)x(mbH) array mbArray containing sample values for a colour component of the current macroblock.

The inverse macroblock scanning process as specified in subclause 6.4.1 is invoked with CurrMbAddr as the input and the output is assigned to ( xO, yO ). During the process in subclause 6.4.1, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.

The sample location ( xP, yP ) is derived by

$$xP = ( xO >> 4 ) * mbW \quad \text{(G-216)}$$
$$yP = ( ( yO >> 4 ) * mbH ) + ( yO \% 2 ) \quad \text{(G-217)}$$

Depending on the variables MbaffFrameFlag and fieldMbFlag, the samples of the array mbArray are derived as follows.

– If MbaffFrameFlag is equal to 1 and fieldMbFlag is equal to 1,

$$mbArray[ x, y ] = picArray[ xP + x, yP + 2 * y ] \quad \text{with } x = 0..(mbW - 1),\ y = 0..(mbH - 1) \tag{G-218}$$

– Otherwise (MbaffFrameFlag is equal to 0 or fieldMbFlag is equal to 0),

$$mbArray[ x, y ] = picArray[ xP + x, yP + y ] \quad \text{with } x = 0..(mbW - 1),\ y = 0..(mbH - 1) \tag{G-219}$$

#### G.8.5.5 Sample array re-initialisation process

Inputs to this process are

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a (PicWidthInSamples$_L$)x(PicHeightInSamples$_L$) array picSamples$_L$ containing luma sample values for the current layer representation,

– when ChromaArrayType is not equal to 0, two (PicWidthInSamples$_C$)x(PicHeightInSamples$_C$) arrays picSamples$_{Cb}$ and picSamples$_{Cr}$ containing chroma sample values for the current layer representation.

Outputs of this process are

– a modified version of the array picSamples$_L$,

– when ChromaArrayType is not equal to 0, modified versions of the arrays picSamples$_{Cb}$ and picSamples$_{Cr}$.

The 16x16 array mbSamples$_L$ is derived by

$$mbSamples_L[ x, y ] = 0 \quad \text{with } x, y = 0..15 \tag{G-220}$$

When ChromaArrayType is not equal to 0, for CX being replaced by Cb and Cr, the (MbWidthC)x(MbHeightC) array mbSamples$_{CX}$ is derived by

$$mbSamples_{CX}[ x, y ] = 0 \quad \text{with } x = 0..(MbWidthC - 1) \text{ and } y = 0..(MbHeightC - 1) \tag{G-221}$$

The picture sample array construction process as specified in subclause G.8.5.4.1 is invoked with fieldMbFlag, mbSamples$_L$, picSamples$_L$, and, when ChromaArrayType is not equal to 0, mbSamples$_{Cb}$, mbSamples$_{Cr}$, picSamples$_{Cb}$, and picSamples$_{Cr}$ as inputs and the outputs are a modified version of picSamples$_L$ and, when ChromaArrayType is not equal to 0, modified versions of picSamples$_{Cb}$ and picSamples$_{Cr}$.

### G.8.6 Resampling processes for prediction data, intra samples, and residual samples

Subclause G.8.6.1 specifies the derivation process for inter-layer predictors for macroblock type, sub-macroblock type, references indices, and motion vectors.

Subclause G.8.6.2 specifies the resampling process for intra samples.

Subclause G.8.6.3 specifies the resampling process for residual samples.

#### G.8.6.1 Derivation process for inter-layer predictors for macroblock type, sub-macroblock type, reference indices, and motion vectors

This process is only invoked when base_mode_flag is equal to 1 or any motion_prediction_flag_lX[ mbPartIdx ] with X being replaced by 0 and 1 and mbPartIdx = 0..3 is equal to 1.

Inputs to this process are

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying the macroblock types for the macroblocks of the reference layer representation,

– a (RefLayerPicSizeInMbs)x4 array refLayerSubMbType specifying the sub-macroblock types for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4 arrays refLayerPredFlagL0 and refLayerPredFlagL1 specifying prediction utilization flags for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4 arrays refLayerRefIdxL0 and refLayerRefIdxL1 specifying reference indices for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4x4x2 arrays refLayerMvL0 and refLayerMvL1 specifying motion vector components for the macroblocks of the reference layer representation,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is less than 2, the reference picture list refPicList0,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Outputs of this process are

– a variable mbTypeILPred specifying a predictor for the macroblock type of the current macroblock,

– a list subMbTypeILPred with 4 elements specifying predictors for sub-macroblock types of the current macroblock,

– two 2x2 arrays refIdxILPredL0 and refIdxILPredL1 specifying inter-layer predictors for the reference indices of the current macroblock,

– two 4x4x2 arrays mvILPredL0 and mvILPredL1 specifying inter-layer predictors for the motion vector components of the current macroblock.

The derivation process for reference layer partition identifications as specified in subclause G.8.6.1.1 is invoked with fieldMbFlag, refLayerFieldMbFlag, refLayerMbType, and refLayerSubMbType as the inputs and the outputs are a variable intraILPredFlag and, when intraILPredFlag is equal to 0, reference layer partition identifications as a 4x4 array refLayerPartIdc with elements refLayerPartIdc[ x, y ].

When slice_type is equal to EI, the bitstream shall not contain data that result in intraILPredFlag equal to 0.

Depending on intraILPredFlag, the 2x2 arrays refIdxILPredL0 and refIdxILPredL1 and the 4x4x2 array mvILPredL0 and mvILPredL1 are derived as follows.

– If intraILPredFlag is equal to 1, all elements of the 2x2 arrays refIdxILPredL0 and refIdxILPredL1 are set equal to –1 and all elements of the 4x4x2 arrays mvILPredL0 and mvILPredL1 are set equal to 0.

– Otherwise (intraILPredFlag is equal to 0), the derivation process for inter-layer predictors for reference indices and motion vectors as specified in subclause G.8.6.1.2 is invoked with fieldMbFlag, refLayerFieldMbFlag, refLayerPredFlagL0, refLayerPredFlagL1, refLayerRefIdxL0, refLayerRefIdxL1, refLayerMvL0, refLayerMvL1, refLayerPartIdc, refPicList0 (when available), and refPicList1 (when available) as the inputs and the outputs are the arrays refIdxILPredL0, refIdxILPredL1, mvILPredL0, and mvILPredL1.

Depending on intraILPredFlag, the variable mbTypeILPred and the list subMbTypeILPred are derived as follows.

– If intraILPredFlag is equal to 1, all elements subMbTypeILPred[ mbPartIdx ] of the list subMbTypeILPred with mbPartIdx = 0..3 are marked as unspecified, and the variable mbTypeILPred is derived as follows.

  – If tcoeff_level_prediction_flag is equal to 1, mbTypeILPred is set equal to refLayerMbType[ CurrMbAddr ].

  – Otherwise (tcoeff_level_prediction_flag is equal to 0), mbTypeILPred is set equal to I_BL.

– Otherwise (intraILPredFlag is equal to 0), the derivation process for inter-layer predictors for P and B macroblock and sub-macroblock types as specified in subclause G.8.6.1.3 is invoked with refIdxILPredL0, refIdxILPredL1, mvILPredL0, and mvILPredL1 as the inputs and the outputs are the variable mbTypeILPred and the list subMbTypeILPred.

**G.8.6.1.1 Derivation process for reference layer partition identifications**

Inputs to this process are

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying the macroblock types for the macroblocks of the reference layer representation,

– a (RefLayerPicSizeInMbs)x4 array refLayerSubMbType specifying the sub-macroblock types for the macroblocks of the reference layer representation.

Outputs of this process are

– a variable intraILPredFlag specifying whether the current macroblock can be predicted by inter-layer intra prediction or, in the case of tcoeff_level_prediction_flag equal to 1, by a combination of intra-layer intra prediction and inter-layer prediction,

– when intraILPredFlag is equal to 0, reference layer partition identifications for the current macroblock as a 4x4 array refLayerPartIdc with elements refLayerPartIdc[ x, y ].

When the 4x4 array refLayerPartIdc is output of this process, each of its elements refLayerPartIdc[ x, y ] specifies the macroblock address, the macroblock partition index, and the sub-macroblock partition index of the partition in the reference layer representation that can be used for inter-layer motion prediction of the macroblock or sub-macroblock partition of the current macroblock that contains the 4x4 block with coordinates x and y.

For each 4x4 block with block coordinates x, y = 0..3, the element refLayerPartIdc[ x, y ] of the 4x4 array refLayerPartIdc is derived by applying the following ordered steps.

1. The derivation process for reference layer partitions as specified in subclause G.6.2 is invoked with the luma location ( 4 * x + 1, 4 * y + 1 ), fieldMbFlag, refLayerFieldMbFlag, refLayerMbType, and refLayerSubMbType as the inputs and the outputs are a macroblock address refMbAddr, a macroblock partition index refMbPartIdx, and a sub-macroblock partition index refSubMbPartIdx of a partition in the reference layer representation.

   The bitstream shall not contain data that result in refMbAddr, refMbPartIdx, or refSubMbPartIdx being marked as not available.

2. The element refLayerPartIdc[ x, y ] of the array refLayerPartIdc is derived as follows.

   – If refLayerMbType[ refMbAddr ] is equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL, refLayerPartIdc[ x, y ] is set equal to –1.

   – Otherwise (refLayerMbType[ refMbAddr ] is not equal to I_PCM, I_16x16, I_8x8, I_4x4, or I_BL), refLayerPartIdc[ x, y ] is derived by

   $$refLayerPartIdc[ x, y ] = 16 * refMbAddr + 4 * refMbPartIdx + refSubMbPartIdx \quad (G\text{-}222)$$

The variable intraILPredFlag is derived as follows.

– If all elements refLayerPartIdc[ x, y ] with x, y = 0..3 are equal to –1, intraILPredFlag is set equal to 1.

– Otherwise (any element refLayerPartIdc[ x, y ] with x, y = 0..3 is not equal to –1, intraILPredFlag is set equal to 0.

When intraILPredFlag is equal to 0 and RestrictedSpatialResolutionChangeFlag is equal to 0, the 4x4 array refLayerPartIdc is modified by the following ordered steps.

NOTE – By the following process the elements refLayerPartIdc[ x, y ] that are equal to –1 are replaced by elements refLayerPartIdc[ x, y ] that are not equal to –1. This process can also be applied when RestrictedSpatialResolutionChangeFlag is equal to 1 or intraILPredFlag is equal to 1, but in this case, the 4x4 array refLayerPartIdc is not modified.

1. For each 8x8 block with block coordinates xP, yP = 0..1, the following ordered steps are specified.

   a. The variables xO and yO are set equal to (2 * xP) and (2 * yP), respectively.

   b. All elements procI4x4Blk[ xS, yS ] of the 2x2 array procI4x4Blk with xS, yS = 0..1 are set equal to 0.

   c. The 4x4 blocks of the current 8x8 block with block coordinates xS, yS = 0..1 are processed in increasing order of (2 * yS + xS), and when refLayerPartIdc[ xO + xS, yO + yS ] is equal to –1 for a 4x4 block, the element procI4x4Blk[ xS, yS ] of the array procI4x4Blk is set equal to 1 and the following applies.

      – If procI4x4Blk[ 1 – xS, yS ] is equal to 0 and refLayerPartIdc[ xO + 1 – xS, yO + yS ] is not equal to –1, the element refLayerPartIdc[ xO + xS, yO + yS ] is modified by

      $$refLayerPartIdc[ xO + xS, yO + yS ] = refLayerPartIdc[ xO + 1 – xS, yO + yS ] \quad (G\text{-}223)$$

– Otherwise, if procI4x4Blk[ xS, 1 – yS ] is equal to 0 and refLayerPartIdc[ xO + xS, yO + 1 – yS ] is not equal to –1, the element refLayerPartIdc[ xO + xS, yO + yS ] is modified by

refLayerPartIdc[ xO + xS, yO + yS ] = refLayerPartIdc[ xO + xS, yO + 1 – yS ] (G-224)

– Otherwise, if procI4x4Blk[ 1 – xS, 1 – yS ] is equal to 0 and refLayerPartIdc[ xO + 1 - xS, yO + 1 - yS ] is not equal to –1, the element refLayerPartIdc[ xO + xS, yO + yS ] is modified by

refLayerPartIdc[ xO + xS, yO + yS ] = refLayerPartIdc[ xO + 1 – xS, yO + 1 – yS ] (G-225)

– Otherwise, the element refLayerPartIdc[ xO + xS, yO + yS ] is not modified.

2. All elements procI8x8Blk[ xP, yP ] of the 2x2 array procI8x8Blk with xP, yP = 0..1 are set equal to 0.

3. The 8x8 blocks with block coordinates xP, yP = 0..1 are processed in increasing order of (2 * yP + xP), and when refLayerPartIdc[ 2 * xP, 2 * yP ] is equal to –1 for an 8x8 block, the element procI8x8Blk[ xP, yP ] of the array procI8x8Blk is set equal to 1 and the following applies.

– If procI8x8Blk[ 1 – xP, yP ] is equal to 0 and refLayerPartIdc[ 2 – xP, 2 * yP ] is not equal to –1, the elements refLayerPartIdc[ 2 * xP + xS, 2 * yP + yS ] with xS, yS = 0..1 are modified by

refLayerPartIdc[ 2 * xP + xS, 2 * yP + yS ] = refLayerPartIdc[ 2 – xP, 2 * yP + yS ] (G-226)

– Otherwise, if procI8x8Blk [ xP, 1 – yP ] is equal to 0 and refLayerPartIdc[ 2 * xP, 2 – yP ] is not equal to –1, the elements refLayerPartIdc[ 2 * xP + xS, 2 * yP + yS ] with xS, yS = 0..1 are modified by

refLayerPartIdc[ 2 * xP + xS, 2 * yP + yS ] = refLayerPartIdc[ 2 * xP + xS, 2 – yP ] (G-227)

– Otherwise, if procI8x8Blk[ 1 – xP, 1 – yP ] is equal to 0 and refLayerPartIdc [ 2 – xP, 2 – yP ] is not equal to –1, the elements refLayerPartIdc[ 2 * xP + xS, 2 * yP + yS ] with xS, yS = 0..1 are modified by

refLayerPartIdc[ 2 * xP + xS, 2 * yP + yS ] = refLayerPartIdc[ 2 – xP, 2 – yP ] (G-228)

– Otherwise, the elements refLayerPartIdc[ 2 * xP + xS, 2 * yP + yS ] with xS, yS = 0..1 are not modified.

**G.8.6.1.2 Derivation process for inter-layer predictors for reference indices and motion vectors**

Inputs to this process are

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– two (RefLayerPicSizeInMbs)x4 arrays refLayerPredFlagL0 and refLayerPredFlagL1 specifying prediction utilization flags for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4 arrays refLayerRefIdxL0 and refLayerRefIdxL1 specifying reference indices for the macroblocks of the reference layer representation,

– two (RefLayerPicSizeInMbs)x4x4x2 arrays refLayerMvL0 and refLayerMvL1 specifying motion vector components for the macroblocks of the reference layer representation,

– a 4x4 array refLayerPartIdc specifying reference layer partition identifications for the 4x4 blocks of the current macroblock,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is less than 2, the reference picture list refPicList0,

– when CroppingChangeFlag is equal to 1 and (slice_type % 5) is equal to 1, the reference picture list refPicList1.

Outputs of this process are

– two 2x2 arrays refIdxILPredL0 and refIdxILPredL0 specifying inter-layer predictors for the reference indices of the current macroblock,

– two 4x4x2 arrays mvILPredL0 and mvILPredL0 specifying inter-layer predictors for the motion vector components of the current macroblock.

Let tempRefIdxPredL0 and tempRefIdxPredL1 be two 4x4 arrays with elements tempRefIdxPredL0[ x, y ] and tempRefIdxPredL1[ x, y ], respectively, that specify auxiliary inter-layer predictors for reference indices.

For each 4x4 block indexed by x, y = 0..3 and for X being replaced by 0 and 1, the auxiliary reference index predictor tempRefIdxPredLX[ x, y ] and the motion vector predictor mvILPredLX[ x, y ] are derived as follows.

– If refLayerPredFlagLX[ refLayerPartIdc[ x, y ] / 16 ][ ( refLayerPartIdc[ x, y ] % 16 ) / 4 ] is equal 0, the reference index predictor tempRefIdxPredLX[ x, y ] and the motion vector predictor mvILPredLX[ x, y ] are derived by

tempRefIdxPredLX[ x, y ] = –1 (G-229)
mvILPredLX[ x, y ][ 0 ] = 0 (G-230)
mvILPredLX[ x, y ][ 1 ] = 0 (G-231)

– Otherwise (refLayerPredFlagLX[ refLayerPartIdc[ x, y ] / 16 ][ ( refLayerPartIdc[ x, y ] % 16 ) / 4 ] is equal to 1), the following ordered steps are specified.

1. The variables refMbAddr, refMbPartIdx, and refSubMbPartIdx are derived by

refMbAddr = refLayerPartIdc[ x, y ] / 16 (G-232)
refMbPartIdx = ( refLayerPartIdc[ x, y ] % 16 ) / 4 (G-233)
refSubMbPartIdx = refLayerPartIdc[ x, y ] % 4 (G-234)

2. The auxiliary reference index predictor tempRefIdxPredLX[ x, y ] is derived by

tempRefIdxPredLX[ x, y ] = refLayerRefIdxLX[ refMbAddr ][ refMbPartIdx ]
* ( 1 + fieldMbFlag – field_pic_flag )
/ ( 1 + refLayerFieldMbFlag[ refMbAddr ] – RefLayerFieldPicFlag ) (G-235)

3. The motion vector aMv is set equal to refLayerMvLX[ refMbAddr ][ refMbPartIdx ][ refSubMbPartIdx ], and afterwards its vertical component aMv[ 1 ] is modified by

aMv[ 1 ] = aMv[ 1 ] * ( 1 + refLayerFieldMbFlag[ refMbAddr ] ) (G-236)

4. The variables scaledW, scaledH, refLayerW, and refLayerH are derived by

scaledW = $ScaledRefLayerPicWidthInSamples_L$ (G-237)
scaledH = $ScaledRefLayerPicHeightInSamples_L$ * ( 1 + field_pic_flag ) (G-238)
refLayerW = $RefLayerPicWidthInSamples_L$ (G-239)
refLayerH = $RefLayerPicHeightInSamples_L$ * ( 1 + RefLayerFieldPicFlag ) (G-240)

5. The variables dOX, dOY, dSW, and dSH are derived as follows.

– If CroppingChangeFlag is equal to 0 or the reference picture refPicListX[ tempRefIdxPredLX[ x, y ] ] is not available, dOX, dOY, dSW, and dSH are set equal to 0.

– Otherwise (CroppingChangeFlag is equal to 1 and the reference picture refPicListX[ tempRefIdxPredLX[ x, y ] ] is available), the variables refPicScaledRefLayerLeftOffset, refPicScaledRefLayerRightOffset, refPicScaledRefLayerTopOffset, and refPicScaledRefLayerBottomOffset are set equal to the variables ScaledRefLayerLeftOffset, ScaledRefLayerRightOffset, ScaledRefLayerTopOffset, and ScaledRefLayerBottomOffset, respectively, that are associated with the layer representation of the reference picture refPicListX[ tempRefIdxPredLX[ x, y ] ] that has the same value of DQId as the current layer representation, and the variables dOX, dOY, dSW, and dSH are derived by

dOX = ScaledRefLayerLeftOffset – refPicScaledRefLayerLeftOffset (G-241)
dOY = ScaledRefLayerTopOffset – refPicScaledRefLayerTopOffset (G-242)
dSW = ScaledRefLayerRightOffset – refPicScaledRefLayerRightOffset + dOX (G-243)
dSH = ScaledRefLayerBottomOffset – refPicScaledRefLayerBottomOffset + dOY (G-244)

6. The variables scaleX and scaleY are derived by

$$scaleX = ( ( ( scaledW + dSW ) << 16 ) + ( refLayerW >> 1 ) ) / refLayerW \quad \text{(G-245)}$$
$$scaleY = ( ( ( scaledH + dSH ) << 16 ) + ( refLayerH >> 1 ) ) / refLayerH \quad \text{(G-246)}$$

7. The motion vector aMv is scaled by

$$aMv[ 0 ] = ( aMv[ 0 ] * scaleX + 32768 ) >> 16 \quad \text{(G-247)}$$
$$aMv[ 1 ] = ( aMv[ 1 ] * scaleY + 32768 ) >> 16 \quad \text{(G-248)}$$

8. When CroppingChangeFlag is equal to 1, the motion vector aMv is modified by applying the following ordered steps.

   a. The inverse macroblock scanning process as specified in subclause 6.4.1 is invoked with CurrMbAddr as the input and the output is a luma location ( xMbPic, yMbPic ). For this invocation of the process specified in subclause 6.4.1, the current macroblock is treated as field macroblock when fieldMbFlag is equal to 1, and it is treated as frame macroblock when fieldMbFlag is equal to 0.

   b. The luma location ( xFrm, yFrm ) is derived by

$$xFrm = ( xMbPic + ( 4 * x + 1 ) ) \quad \text{(G-249)}$$
$$yFrm = ( yMbPic + ( 4 * y + 1 ) * ( 1 + fieldMbFlag – field_pic_flag ) ) * ( 1 + field_pic_flag ) \quad \text{(G-250)}$$

   c. The variables scaleX and scaleY are modified by

$$scaleX = ( ( ( 4 * dSW ) << 16 ) + ( scaledW >> 1 ) ) / scaledW \quad \text{(G-251)}$$
$$scaleY = ( ( ( 4 * dSH ) << 16 ) + ( scaledH >> 1 ) ) / scaledH \quad \text{(G-252)}$$

   d. The motion vector aMv is modified by

$$aMv[ 0 ] += ( ( ( xFrm – ScaledRefLayerLeftOffset ) * scaleX + 32768 ) >> 16 ) – 4 * dOX \quad \text{(G-253)}$$
$$aMv[ 1 ] += ( ( ( yFrm – ScaledRefLayerTopOffset ) * scaleY + 32768 ) >> 16 ) – 4 * dOY \quad \text{(G-254)}$$

9. The motion vector predictor mvILPredLX[ x, y ] is derived by

$$mvILPredLX[ x, y ][ 0 ] = aMv[ 0 ] \quad \text{(G-255)}$$
$$mvILPredLX[ x, y ][ 1 ] = aMv[ 1 ] / ( 1 + fieldMbFlag ) \quad \text{(G-256)}$$

For each 8x8 block indexed by xP, yP = 0..1 and for X being replaced by 0 or 1, the reference index predictor refIdxILPredLX[ xP, yP ] is set equal to tempRefIdxPredLX[ 2 * xP, 2 * yP ], and when RestrictedSpatialResolutionChangeFlag is equal to 0, the following ordered steps are specified.

NOTE – The following process can also be applied when RestrictedSpatialResolutionChangeFlag is equal to 1, but in this case, the reference index predictor refIdxILPredLX[ xP, yP ] and the motion vector predictors mvILPredLX[ 2 * xP + xS, 2 * yP + yS ] with xS, yS = 0..1 will not be modified.

1. The 4x4 blocks indexed by xS, yS = 0..1 of the current 8x8 block are processed in increasing order of (2 * yS + xS), and for each 4x4 block, the reference index predictor refIdxILPredLX[ xP, yP ] is modified by

$$refIdxILPredLX[ xP, yP ] = MinPositive( refIdxILPredLX[ xP, yP ], tempRefIdxPredLX[ 2 * xP + xS, 2 * yP + yS ] ) \quad \text{(G-257)}$$

   with

$$MinPositive( a, b ) = \begin{cases} Min( a, b ) & \text{if } a >= 0 \text{ and } b >= 0 \\ Max( a, b ) & \text{otherwise} \end{cases} \quad \text{(G-258)}$$

2. The 4x4 blocks index by xS, yS = 0..1 of the current 8x8 block are processed in increasing order of (2 * yS + xS), and for each 4x4 block, when tempRefIdxPredLX[ 2 * xP + xS, 2 * yP + yS ] is not equal to the reference index predictor refIdxILPredLX[ xP, yP ], the following applies.

– If tempRefIdxPredLX[ 2 * xP + 1 – xS, 2 * yP + yS ] is equal to refIdxILPredLX[ xP, yP ], the motion vector predictor mvILPredLX[ 2 * xP + xS, 2 * yP + yS ] is modified by

$$mvILPredLX[ 2 * xP + xS, 2 * yP + yS ] = mvILPredLX[ 2 * xP + 1 - xS, 2 * yP + yS ] \quad \text{(G-259)}$$

– Otherwise, if tempRefIdxPredLX[ 2 * xP + xS, 2 * yP + 1 – yS ] is equal to refIdxILPredLX[ xP, yP ], the motion vector predictor mvILPredLX[ 2 * xP + xS, 2 * yP + yS ] is modified by

$$mvILPredLX[ 2 * xP + xS, 2 * yP + yS ] = mvILPredLX[ 2 * xP + xS, 2 * yP + 1 - yS ] \quad \text{(G-260)}$$

– Otherwise (tempRefIdxPredLX[ 2 * xP + 1 – xS, 2 * yP + 1 – yS ] is equal to refIdxILPredLX[ xP, yP ]), the motion vector predictor mvILPredLX[ 2 * xP + xS, 2 * yP + yS ] is modified by

$$mvILPredLX[ 2 * xP + xS, 2 * yP + yS ] = mvILPredLX[ 2 * xP + 1 - xS, 2 * yP + 1 - yS ] \quad \text{(G-261)}$$

When RestrictedSpatialResolutionChangeFlag is equal to 0, for each 8x8 block indexed by xP, yP = 0..1, the motion vector predictor arrays mvILPredL0 and mvILPredL1 are modified by applying the following ordered steps.

NOTE – The following process shall not be applied when RestrictedSpatialResolutionChangeFlag is equal to 1.

1. The variable maxX is derived as follows.
   – If slice_type is equal to EB, maxX is set equal to 1.
   – Otherwise (slice_type is equal to EP), maxX is set equal to 0.
2. The variables xO and yO are set equal to (2 * xP) and (2 * yP), respectively.
3. The function mvDiff( $mv_1$, $mv_2$ ) of two motion vectors $mv_1$ and $mv_2$ is defined by

$$mvDiff( mv_1, mv_2 ) = Abs( mv_1[ 0 ] - mv_2[ 0 ] ) + Abs( mv_1[ 1 ] - mv_2[ 1 ] ) \quad \text{(G-262)}$$

4. The variable subPartSize is derived as follows.
   – If for X = 0..maxX, all of the following conditions are true, subPartSize is set equal to 8x8.
     – mvDiff( mvILPredLX[ xO, yO ], mvILPredLX[ xO + 1, yO ] ) is less than or equal to 1
     – mvDiff( mvILPredLX[ xO, yO ], mvILPredLX[ xO, yO + 1 ] ) is less than or equal to 1
     – mvDiff( mvILPredLX[ xO, yO ], mvILPredLX[ xO + 1, yO + 1 ] ) is less than or equal to 1
   – Otherwise, if for X = 0..maxX, all of the following conditions are true, subPartSize is set equal to 8x4.
     – mvDiff( mvILPredLX[ xO, yO ], mvILPredLX[ xO + 1, yO ] ) is less than or equal to 1
     – mvDiff( mvILPredLX[ xO, yO + 1 ], mvILPredLX[ xO + 1, yO + 1 ] ) is less than or equal to 1
   – Otherwise, if for X = 0..maxX, all of the following conditions are true, subPartSize is set equal to 4x8.
     – mvDiff( mvILPredLX[ xO, yO ], mvILPredLX[ xO, yO + 1 ] ) is less than or equal to 1
     – mvDiff( mvILPredLX[ xO + 1, yO ], mvILPredLX[ xO + 1, yO + 1 ] ) is less than or equal to 1
   – Otherwise, subPartSize is set equal to 4x4.
5. When subPartSize is not equal to 4x4, for X = 0..maxX, the motion vectors tempMvALX and tempMvBLX (when subPartSize is equal to 8x4 or 4x8) are derived as follows
   – If subPartSize is equal to 8x8, tempMvALX is derived by

$$\begin{aligned} tempMvALX[ c ] = ( & mvILPredLX[ xO, \ yO \ ][ c ] + \\ & mvILPredLX[ xO + 1, yO \ ][ c ] + \\ & mvILPredLX[ xO, \ yO + 1 ][ c ] + \\ & mvILPredLX[ xO + 1, yO + 1 ][ c ] + 2 ) >> 2 \quad \text{with } c = 0..1 \end{aligned} \quad \text{(G-263)}$$

   – Otherwise, if subPartSize is equal to 8x4, tempMvALX and tempMvBLX are derived by

$$tempMvALX[\,c\,] = (\,mvILPredLX[\,xO,\ yO\,][\,c\,] + mvILPredLX[\,xO+1,\ yO\,][\,c\,]+1\,) >> 1 \quad \text{with } c = 0..1 \qquad \text{(G-264)}$$

$$tempMvBLX[\,c\,] = (\,mvILPredLX[\,xO,\ yO+1\,][\,c\,] + mvILPredLX[\,xO+1,\ yO+1\,][\,c\,]+1\,) >> 1 \quad \text{with } c = 0..1 \qquad \text{(G-265)}$$

– Otherwise (subPartSize is equal to 4x8), tempMvALX and tempMvBLX are derived by

$$tempMvALX[\,c\,] = (\,mvILPredLX[\,xO,\ yO\,][\,c\,] + mvILPredLX[\,xO,\ yO+1\,][\,c\,]+1\,) >> 1 \quad \text{with } c = 0..1 \qquad \text{(G-266)}$$

$$tempMvBLX[\,c\,] = (\,mvILPredLX[\,xO+1,\ yO\,][\,c\,] + mvILPredLX[\,xO+1,\ yO+1\,][\,c\,]+1\,) >> 1 \quad \text{with } c = 0..1 \qquad \text{(G-267)}$$

6. When subPartSize is not equal to 4x4, for X = 0..maxX, the motion vector predictor array mvILPredLX is modified as follows.

– If subPartSize is equal to 8x8, the array mvILPredLX is modified by

$$mvILPredLX[\,xO+xS,\ yO+yS\,][\,c\,] = tempMvALX[\,c\,] \quad \text{with } xS, yS = 0..1 \text{ and } c = 0..1 \qquad \text{(G-268)}$$

– Otherwise, if subPartSize is equal to 8x4, the array mvILPredLX is modified by

$$mvILPredLX[\,xO+xS,\ yO\,][\,c\,] = tempMvALX[\,c\,] \quad \text{with } xS = 0..1 \text{ and } c = 0..1 \qquad \text{(G-269)}$$
$$mvILPredLX[\,xO+xS,\ yO+1\,][\,c\,] = tempMvBLX[\,c\,] \quad \text{with } xS = 0..1 \text{ and } c = 0..1 \qquad \text{(G-270)}$$

– Otherwise (subPartSize is equal to 4x8), the array mvILPredLX is modified by

$$mvILPredLX[\,xO,\ yO+yS\,][\,c\,] = tempMvALX[\,c\,] \quad \text{with } yS = 0..1 \text{ and } c = 0..1 \qquad \text{(G-271)}$$
$$mvILPredLX[\,xO+1,\ yO+yS\,][\,c\,] = tempMvBLX[\,c\,] \quad \text{with } yS = 0..1 \text{ and } c = 0..1 \qquad \text{(G-272)}$$

#### G.8.6.1.3 Derivation process for inter-layer predictors for P and B macroblock and sub-macroblock types

This process is only invoked when slice_type is equal to EP or EB.

Inputs to this process are

– two 2x2 arrays refIdxILPredL0 and refIdxILPredL0 specifying predictors for the reference indices of the current macroblock,

– two 4x4x2 arrays mvILPredL0 and mvILPredL0 specifying predictors for the motion vectors of the current macroblock.

Outputs of this process are

– a variable mbTypeILPred specifying a predictor for the macroblock type of the current macroblock,

– a list subMbTypeILPred with 4 elements specifying predictors for sub-macroblock types of the current macroblock.

The variable maxX is derived as follows.

– If slice_type is equal to EB, maxX is set equal to 1.

– Otherwise (slice_type is equal to EP), maxX is set equal to 0.

The macroblock type predictor mbTypeILPred is derived by applying the following ordered steps.

1. The variable partitionSize is derived as follows.

– If for X = 0..maxX, all of the following conditions are true, partitionSize is set equal to 16x16.

– all elements refIdxILPredLX[ x, y ] with x, y = 0..1 are the same

– all elements mvILPredLX[ x, y ] with x, y = 0..3 are the same

– Otherwise, if for X = 0..maxX, all of the following conditions are true, partitionSize is set equal to 16x8.

- refIdxILPredLX[ 0, 0 ] is equal to refIdxILPredLX[ 1, 0 ]
- refIdxILPredLX[ 0, 1 ] is equal to refIdxILPredLX[ 1, 1 ]
- all elements mvILPredLX[ x, y ] with x = 0..3 and y = 0..1 are the same
- all elements mvILPredLX[ x, y ] with x = 0..3 and y = 2..3 are the same

– Otherwise, if for X = 0..maxX, all of the following conditions are true, partitionSize is set equal to 8x16.

- refIdxILPredLX[ 0, 0 ] is equal to refIdxILPredLX[ 0, 1 ]
- refIdxILPredLX[ 1, 0 ] is equal to refIdxILPredLX[ 1, 1 ]
- all elements mvILPredLX[ x, y ] with x = 0..1 and y = 0..3 are the same
- all elements mvILPredLX[ x, y ] with x = 2..3 and y = 0..3 are the same

– Otherwise, partitionSize is set equal to 8x8.

2. When slice_type is equal to EB and partitionSize is not equal to 8x8, the variable partPredModeA is derived by

$$\begin{aligned} partPredModeA = {} & ( refIdxILPredL1[ 0, 0 ] >= 0 \ ? \ 2 \ : \ 0 ) + \\ & ( refIdxILPredL0[ 0, 0 ] >= 0 \ ? \ 1 \ : \ 0 ) \end{aligned} \tag{G-273}$$

3. When slice_type is equal to EB and partitionSize is equal to 16x8 or 8x16, the variable partPredModeB is derived by

$$\begin{aligned} partPredModeB = {} & ( refIdxILPredL1[ 1, 1 ] >= 0 \ ? \ 2 \ : \ 0 ) + \\ & ( refIdxILPredL0[ 1, 1 ] >= 0 \ ? \ 1 \ : \ 0 ) \end{aligned} \tag{G-274}$$

4. Depending on slice_type, partitionSize, partPredModeA (when applicable), and partPredModeB (when applicable), the macroblock type predictor mbTypeILPred is derived as specified in Table G-6.

All elements subMbTypeILPred[ mbPartIdx ] of the list subMbTypeILPred with mbPartIdx = 0..3 are marked as "unspecified".

When mbTypeILPred is equal to P_8x8 or B_8x8, each element subMbTypeILPred[ mbPartIdx ] with mbPartIdx = 0..3 is modified by applying the following ordered steps.

1. The coordinate offset ( xO, yO ) is set equal to ( 2 * ( mbPartIdx % 2 ), 2 * ( mbPartIdx / 2 ) ).
2. The variable subPartitionSize is derived as follows.
   - If for X = 0..maxX, all elements mvILPredLX[ xO + xS, yO + yS ] with xS, yS = 0..1 are the same, subPartitionSize is set equal to 8x8.
   - Otherwise, if for X = 0..maxX, mvILPredLX[ xO, yO ] is equal to mvILPredLX[ xO + 1, yO ] and mvILPredLX[ xO, yO + 1 ] is equal to mvILPredLX[ xO + 1, yO + 1 ], subPartitionSize is set equal to 8x4.
   - Otherwise, if for X = 0..maxX, mvILPredLX[ xO, yO ] is equal to mvILPredLX[ xO, yO + 1 ] and mvILPredLX[ xO + 1, yO ] is equal to mvILPredLX[ xO + 1, yO + 1 ], subPartitionSize is set equal to 4x8.
   - Otherwise, subPartitionSize is set equal to 4x4.
3. When slice_type is equal to EB, the variable partPredMode is derived by

$$\begin{aligned} partPredMode = {} & ( refIdxILPredL1[ xO / 2, yO / 2 ] >= 0 \ ? \ 2 \ : \ 0 ) + \\ & ( refIdxILPredL0[ xO / 2, yO / 2 ] >= 0 \ ? \ 1 \ : \ 0 ) \end{aligned} \tag{G-275}$$

4. Depending on slice_type, subPartitionSize, and partPredMode (when applicable), the sub-macroblock type predictor subMbTypeILPred[ mbPartIdx ] is derived as specified in Table G-7.

**Table G-6 – Macroblock type predictors mbTypeILPred**

| slice_type | partitionSize | partPredModeA | partPredModeB | mbTypeILPred | slice_type | partitionSize | partPredModeA | partPredModeB | mbTypeILPred |
|---|---|---|---|---|---|---|---|---|---|
| EB | 16x16 | 1 | na | B_L0_16x16 | EB | 16x8 | 2 | 3 | B_L1_Bi_16x8 |
| EB | 16x16 | 2 | na | B_L1_16x16 | EB | 8x16 | 2 | 3 | B_L1_Bi_8x16 |
| EB | 16x16 | 3 | na | B_Bi_16x16 | EB | 16x8 | 3 | 1 | B_Bi_L0_16x8 |
| EB | 16x8 | 1 | 1 | B_L0_L0_16x8 | EB | 8x16 | 3 | 1 | B_Bi_L0_8x16 |
| EB | 8x16 | 1 | 1 | B_L0_L0_8x16 | EB | 16x8 | 3 | 2 | B_Bi_L1_16x8 |
| EB | 16x8 | 2 | 2 | B_L1_L1_16x8 | EB | 8x16 | 3 | 2 | B_Bi_L1_8x16 |
| EB | 8x16 | 2 | 2 | B_L1_L1_8x16 | EB | 16x8 | 3 | 3 | B_Bi_Bi_16x8 |
| EB | 16x8 | 1 | 2 | B_L0_L1_16x8 | EB | 8x16 | 3 | 3 | B_Bi_Bi_8x16 |
| EB | 8x16 | 1 | 2 | B_L0_L1_8x16 | EB | 8x8 | na | na | B_8x8 |
| EB | 16x8 | 2 | 1 | B_L1_L0_16x8 | EP | 16x16 | na | na | P_L0_16x16 |
| EB | 8x16 | 2 | 1 | B_L1_L0_8x16 | EP | 16x8 | na | na | P_L0_L0_16x8 |
| EB | 16x8 | 1 | 3 | B_L0_Bi_16x8 | EP | 8x16 | na | na | P_L0_L0_8x16 |
| EB | 8x16 | 1 | 3 | B_L0_Bi_8x16 | EP | 8x8 | na | na | P_8x8 |

**Table G-7 – Sub-macroblock type predictors subMbTypeILPred[ mbPartIdx ]**

| slice_type | subPartitionSize | partPredMode | subMbTypeILPred [ mbPartIdx ] | slice_type | subPartitionSize | partPredMode | subMbTypeILPred [ mbPartIdx ] |
|---|---|---|---|---|---|---|---|
| EB | 8x8 | 1 | B_L0_8x8 | EB | 4x8 | 3 | B_Bi_4x8 |
| EB | 8x8 | 2 | B_L1_8x8 | EB | 4x4 | 1 | B_L0_4x4 |
| EB | 8x8 | 3 | B_Bi_8x8 | EB | 4x4 | 2 | B_L1_4x4 |
| EB | 8x4 | 1 | B_L0_8x4 | EB | 4x4 | 3 | B_Bi_4x4 |
| EB | 8x4 | 2 | B_L1_8x4 | EP | 8x8 | na | P_L0_8x8 |
| EB | 8x4 | 3 | B_Bi_8x4 | EP | 8x4 | na | P_L0_8x4 |
| EB | 4x8 | 1 | B_L0_4x8 | EP | 4x8 | na | P_L0_4x8 |
| EB | 4x8 | 2 | B_L1_4x8 | EP | 4x4 | na | P_L0_4x4 |

### G.8.6.2 Resampling process for intra samples

Inputs to this process are

- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a one-dimensional array refLayerSliceIdc with RefLayerPicSizeInMbs elements specifying slice identifications for the macroblocks of the reference layer representation,
- a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,
- a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation,
- a (RefLayerPicWidthInSamples$_L$)x(RefLayerPicHeightInSamples$_L$) array refLayerPicSamples$_L$ of luma samples for the reference layer representation,
- a (PicWidthInSamples$_L$)x(PicHeightInSamples$_L$) array picSamples$_L$ of luma samples,
- when ChromaArrayType is not equal to 0, two (RefLayerPicWidthInSamples$_C$)x(RefLayerPicHeightInSamples$_C$) arrays refLayerPicSamples$_{Cb}$ and refLayerPicSamples$_{Cr}$ of chroma samples for the reference layer representation,
- when ChromaArrayType is not equal to 0, two (PicWidthInSamples$_C$)x(PicHeightInSamples$_C$) arrays picSamples$_{Cb}$ and picSamples$_{Cr}$ of chroma samples.

Outputs of this process are

- a modified version of the array picSamples$_L$ of luma samples,
- when ChromaArrayType is not equal to 0, modified versions of the arrays picSamples$_{Cb}$ and picSamples$_{Cr}$ of chroma samples.

The resampling process for intra samples of a macroblock colour component as specified in subclause G.8.6.2.1 is invoked with chromaFlag equal to 0, mbW equal to 16, mbH equal to 16, fieldMbFlag, refLayerPicSamples$_L$, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the output is the 16x16 array mbPred$_L$ of Intra_Base prediction samples for the luma component of the current macroblock.

When ChromaArrayType is not equal to 0, for CX being replaced by Cb and Cr, the resampling process for intra samples of a macroblock colour component as specified in subclause G.8.6.2.1 is invoked with chromaFlag equal to 1, mbW equal to MbWidthC, mbH equal to MbHeightC, fieldMbFlag, refLayerPicSamples$_{CX}$, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the output is the (MbWidthC)x(MbHeightC) array mbPred$_{CX}$ of Intra_Base prediction samples for the CX component of the current macroblock.

The picture sample array construction process as specified in subclause G.8.5.4.1 is invoked with fieldMbFlag, mbPred$_L$, picSamples$_L$ and, when ChromaArrayType is not equal to 0, mbPred$_{Cb}$, mbPred$_{Cr}$, picSamples$_{Cb}$, and picSamples$_{Cr}$ as the inputs and the outputs are a modified version of picSamples$_L$ and, when ChromaArrayType is not equal to 0, modified versions of picSamples$_{Cb}$, and picSamples$_{Cr}$.

#### G.8.6.2.1 Resampling process for intra samples of a macroblock colour component

Inputs to this process are

- a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,
- two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,
- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- an array refLayerPicSamples, which is a (RefLayerPicWidthInSamples$_L$)x(RefLayerPicHeightInSamples$_L$) array containing constructed intra luma sample values for the reference layer representation when chromaFlag is equal to 0 or a (RefLayerPicWidthInSamples$_C$)x(RefLayerPicHeightInSamples$_C$) array containing constructed intra chroma sample values for the reference layer representation when chromaFlag is equal to 1,
- a one-dimensional array refLayerSliceIdc with RefLayerPicSizeInMbs elements specifying slice identifications for the macroblocks of the reference layer representation,
- a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation.

Output of this process is an (mbW)x(mbH) array mbPred of Intra_Base prediction samples.

The variable botFieldFlag is derived as follows.

– If RefLayerFrameMbsOnlyFlag is equal to 0, botFieldFlag is set equal to 0.

– Otherwise, if field_pic_flag is equal to 1, botFieldFlag is set equal to bottom_field_flag.

– Otherwise, if RefLayerFieldPicFlag is equal to 1, botFieldFlag is set equal to RefLayerBottomFieldFlag.

– Otherwise, if fieldMbFlag is equal to 1, botFieldFlag is set equal to (CurrMbAddr % 2).

– Otherwise, botFieldFlag is set equal to 0.

The variable frameBasedResamplingFlag is derived as follows.

– If all of the following conditions are true, frameBasedResamplingFlag is set equal to 1.

  – RefLayerFrameMbsOnlyFlag is equal to 1

  – frame_mbs_only_flag is equal to 1

– Otherwise, frameBasedResamplingFlag is set equal to 0.

The variable topAndBotResamplingFlag is derived as follows.

– If all of the following conditions are true, topAndBotResamplingFlag is set equal to 1.

  – RefLayerFrameMbsOnlyFlag is equal to 0

  – RefLayerFieldPicFlag is equal to 0

  – frame_mbs_only_flag is equal to 0

  – fieldMbFlag is equal to 0

– Otherwise, topAndBotResamplingFlag is set equal to 0.

The variable filteringModeFlag is derived as follows.

– If chromaFlag is equal to 0 or ChromaArrayType is equal to 3, filteringModeFlag is set equal to 0.

– Otherwise (chromaFlag is equal to 1 and ChromaArrayType is not equal to 3), filteringModeFlag is set equal to 1.

The array predArray is derived as specified in the following.

– If frameBasedResamplingFlag is equal to 1 or fieldMbFlag is equal to 1, the following ordered steps are specified.

  1. The reference layer sample array construction process prior to intra resampling as specified in subclause G.8.6.2.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, refLayerPicSamples, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are the variables refArrayW, refArrayH, the array refSampleArray of reference layer sample values, and the variables xOffset and yOffset.

  2. The interpolation process for Intra_Base prediction as specified in subclause G.8.6.2.3 is invoked with filteringModeFlag, chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, fldPrdInFrmMbFlag equal to 0, yBorder equal to 0, refArrayW, refArrayH, refSampleArray, xOffset, and yOffset as the inputs and the output is the (mbW)x(mbH) array mbPred of Intra_Base prediction samples.

– Otherwise, if topAndBotResamplingFlag is equal to 0, the following ordered steps are specified.

  1. The reference layer sample array construction process prior to intra resampling as specified in subclause G.8.6.2.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, refLayerPicSamples, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are the variables refArrayW, refArrayH, the array refSampleArray of reference layer sample values, and the variables xOffset and yOffset.

  2. The variable yBorder is set equal to ( 2 – chromaFlag ).

  3. The interpolation process for Intra_Base prediction as specified in subclause G.8.6.2.3 is invoked with filteringModeFlag, chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, fldPrdInFrmMbFlag equal to 1,

yBorder, refArrayW, refArrayH, refSampleArray, xOffset, and yOffset as the inputs and the output is the (mbW)x(mbH / 2 + 2* yBorder) array fieldPredArray of field prediction samples.

4. The vertical interpolation process for Intra_Base prediction as specified in subclause G.8.6.2.4 is invoked with filteringModeFlag, chromaFlag, mbW, mbH, botFieldFlag, yBorder, and fieldPredArray as the inputs and the output is the (mbW)x(mbH) array mbPred of Intra_Base prediction samples.

– Otherwise (topAndBotResamplingFlag is equal to 1), the following ordered steps are specified.

1. The reference layer sample array construction process prior to intra resampling as specified in subclause G.8.6.2.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 0, refLayerPicSamples, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are the variables refArrayTopW, refArrayTopH, the array refSampleArrayTop of top field reference layer sample values, and the variables xOffsetTop and yOffsetTop.

2. The interpolation process for Intra_Base prediction as specified in subclause G.8.6.2.3 is invoked with filteringModeFlag, chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 0, fldPrdInFrmMbFlag equal to 1, yBorder equal to 0, refArrayTopW, refArrayTopH, refSampleArrayTop, xOffsetTop, and yOffsetTop as the inputs and the output is the (mbW)x(mbH / 2) array topFieldPredArray of top field prediction samples.

3. The reference layer sample array construction process prior to intra resampling as specified in subclause G.8.6.2.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 1, refLayerPicSamples, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are the variables refArrayBotW, refArrayBotH, the array refSampleArrayBot of bottom field reference layer sample values, and the variables xOffsetBot and yOffsetBot.

4. The interpolation process for Intra_Base prediction as specified in subclause G.8.6.2.3 is invoked with filteringModeFlag, chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 1, fldPrdInFrmMbFlag equal to 1, yBorder equal to 0, refArrayBotW, refArrayBotH, refSampleArrayBot, xOffsetBot, and yOffsetBot as the inputs and the output is the (mbW)x(mbH / 2) array botFieldPredArray of bottom field prediction samples.

5. Each sample predArray[ x, y ] with x = 0..(mbW – 1) and y = 0..(mbH – 1) of the array mbPred of Intra_Base prediction samples is derived by

$$\begin{aligned} mbPred[\,x, y\,] = (\,(\,y \,\%\, 2\,) == 0\ ?\ & topFieldPredArray[\,x, y >> 1\,] \\ :\ & botFieldPredArray[\,x, y >> 1\,]\,) \end{aligned} \tag{G-276}$$

**G.8.6.2.2 Reference layer sample array construction process prior to intra resampling**

Inputs to this process are

– a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,

– two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable botFieldFlag specifying whether a top or a bottom field is subject to the resampling process (when RefLayerFrameMbsOnlyFlag is equal to 0 or frame_mbs_only_flag is equal to 0),

– an array refLayerPicSamples, which is a ($RefLayerPicWidthInSamples_L$)x($RefLayerPicHeightInSamples_L$) array containing constructed intra luma sample values for the reference layer representation when chromaFlag is equal to 0 or a ($RefLayerPicWidthInSamples_C$)x($RefLayerPicHeightInSamples_C$) array containing constructed intra chroma sample values for the reference layer representation when chromaFlag is equal to 1,

– a one-dimensional array refLayerSliceIdc with RefLayerPicSizeInMbs elements specifying slice identifications for the macroblocks of the reference layer representation,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation.

Outputs of this process are

– two variables refArrayW and refArrayH specifying the width and height, respectively, of the constructed array of reference layer sample values,

– a (refArrayW)x(refArrayH) array refSampleArray of reference layer sample values,

– two variables xOffset and yOffset specifying the x and y coordinate, respectively, of the reference layer sample location that corresponds to the sample refSampleArray[ 0, 0 ] of the array refSampleArray.

The variables refW, refH, refMbW, refMbH, xOffset, yOffset, refArrayW, refArrayH, xMin, yMin, xMax, yMax, yRefScale, and yRefAdd are derived as specified in the following ordered steps.

1. The derivation process for reference layer sample locations in resampling as specified in subclause G.6.3 is invoked with chromaFlag, the sample location ( 0, 0 ), fieldMbFlag, and botFieldFlag as the inputs and the output is the sample location ( xRefMin16, yRefMin16 ) in units of 1/16-th sample.

2. The derivation process for reference layer sample locations in resampling as specified in subclause G.6.3 is invoked with chromaFlag, the sample location ( mbW – 1, mbH – 1 ), fieldMbFlag, and botFieldFlag as the inputs and the output is the sample location ( xRefMax16, yRefMax16 ) in units of 1/16-th sample.

3. With Z being replaced by L for chromaFlag equal to 0 and C for chromaFlag equal to 1, the variables refW, refH, refMbW, and refMbH are derived by

$$refW = RefLayerPicWidthInSamples_Z \quad \text{(G-277)}$$
$$refH = RefLayerPicHeightInSamples_Z \quad \text{(G-278)}$$
$$refMbW = ( ( chromaFlag == 0 ) ? 16 : RefLayerMbWidthC ) \quad \text{(G-279)}$$
$$refMbH = ( ( chromaFlag == 0 ) ? 16 : RefLayerMbHeightC ) \quad \text{(G-280)}$$

4. The variables xOffset, yOffset, refArrayW, and refArrayH are derived by

$$xOffset = ( ( xRefMin16 >> 8 ) << 4 ) – ( refMbW >> 1 ) \quad \text{(G-281)}$$
$$yOffset = ( ( yRefMin16 >> 8 ) << 4 ) – ( refMbH >> 1 ) \quad \text{(G-282)}$$
$$refArrayW = ( ( ( xRefMax16 + 15 ) >> 8 ) << 4 ) + ( refMbW >> 1 ) – xOffset + 16 \quad \text{(G-283)}$$
$$refArrayH = ( ( ( yRefMax16 + 15 ) >> 8 ) << 4 ) + ( refMbH >> 1 ) – yOffset + 16 \quad \text{(G-284)}$$

NOTE 1 – The derived array size might be larger than the array size that is required by the interpolation process for Intra_Base prediction specified in subclause G.8.6.2.3.

5. The variables xMin, yMin, xMax, and yMax are derived by

$$xMin = ( xRefMin16 >> 4 ) – xOffset \quad \text{(G-285)}$$
$$yMin = ( yRefMin16 >> 4 ) – yOffset \quad \text{(G-286)}$$
$$xMax = ( ( xRefMax16 + 15 ) >> 4 ) – xOffset \quad \text{(G-287)}$$
$$yMax = ( ( yRefMax16 + 15 ) >> 4 ) – yOffset \quad \text{(G-288)}$$

6. The variables yRefScale and yRefAdd are derived as follows.

   – If RefLayerFrameMbsOnlyFlag is equal to 1 or RefLayerFieldMbFlag is equal to 1, yRefScale is set equal to 1 and yRefAdd is set equal to 0.

   – Otherwise (RefLayerFrameMbsOnlyFlag is equal to 0 and RefLayerFieldMbFlag is equal to 0), yRefScale is set equal to 2 and yRefAdd is set equal to botFieldFlag.

The variable refSliceIdcMb is marked as "not available".

Each sample refSampleArray[ x, y ] with x = 0..(refArrayW – 1) and y = 0..(refArrayH – 1) is derived as specified in the following ordered steps.

1. A reference layer sample location ( xRef, yRef ) is derived by

$$xRef = max( 0, min( refW – 1, x + xOffset ) ) \quad \text{(G-289)}$$
$$yRef = yRefScale * max( 0, min( refH / yRefScale – 1, y + yOffset ) ) + yRefAdd \quad \text{(G-290)}$$

2. The derivation process for reference layer slice and intra macroblock identifications as specified in subclause G.8.6.2.2.1 is invoked with the reference layer sample location ( xRef, yRef ), refMbW, refMbH, refLayerSliceIdc, refLayerFieldMbFlag, and refLayerMbType as the inputs and the outputs are the reference layer slice identification refSliceIdc and the variable refIntraMbFlag.

3. When constrained_intra_resampling_flag is equal to 1, refIntraMbFlag is equal to 1, x is greater than or equal to xMin, x is less than or equal to xMax, y is greater than or equal to yMin, and y is less than or equal to yMax, the following applies.

   – If the variable refSliceIdcMb is marked as "not available", refSliceIdcMb is marked as "available" and its value is set equal to refSliceIdc.

– Otherwise (refSliceIdcMb is marked as "available"), the bitstream shall not contain data that result in a reference layer slice identification refSliceIdc that is not equal to refSliceIdcMb.

NOTE 2 – This constraint specifies that a macroblock cannot be coded with base_mode_flag equal to 1 when it covers intra-coded macroblocks of more than one slice in the reference layer representation and constrained_intra_resampling_flag is equal to 1.

4. Depending on refIntraMbFlag, constrained_intra_resampling_flag, and refSliceIdc, the following applies.

   – If any of the following conditions is true, the sample refSampleArray[ x, y ] is marked as "not available for Intra_Base prediction" and its value is set equal to 0.

     – refIntraMbFlag is equal to 0

     – constrained_intra_resampling_flag is equal to 1 and refSliceIdc is not equal to refSliceIdcMb

   – Otherwise, the sample refSampleArray[ x, y ] is marked as "available for Intra_Base prediction" and its value is derived by

$$refSampleArray[ x, y ] = refLayerPicSamples[ xRef, yRef ] \quad \text{(G-291)}$$

The construction process for not available sample values prior to intra resampling as specified in subclause G.8.6.2.2.2 is invoked with refMbW, refMbH, refArrayW, refArrayH, refSampleArray, xOffset, and yOffset as the inputs and the output is a modified version of the sample array refSampleArray.

#### G.8.6.2.2.1 Derivation process for reference layer slice and intra macroblock identifications

Inputs to this process are

– a reference layer sample location ( xRef, yRef ) relative to the upper-left sample of the considered colour component of the reference layer picture,

– two variables refMbW and refMbH specifying the width and height, respectively, of a reference layer macroblock for the considered colour component,

– a one-dimensional array refLayerSliceIdc with RefLayerPicSizeInMbs elements specifying slice identifications for the macroblocks of the reference layer representation,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerMbType with RefLayerPicSizeInMbs elements specifying macroblock types for the macroblocks of the reference layer representation.

Outputs of this process are

– a reference layer slice identification refSliceIdc for the slice that covers the input reference layer sample location,

– a variable refIntraMbFlag specifying whether the reference layer macroblock that covers the input reference layer sample location is intra coded.

The reference layer macroblock address refMbAddr is derived as follows.

– If RefLayerMbaffFrameFlag is equal to 0, the variable refMbAddr is derived by

$$refMbAddr = ( yRef / refMbH ) * RefLayerPicWidthInMbs + ( xRef / refMbW ) \quad \text{(G-292)}$$

– Otherwise (RefLayerMbaffFrameFlag is equal to 1), the variable refMbAddr is derived as specified in the following ordered steps.

1. A variable refMbAddrTop is derived by

$$refMbAddrTop = 2 * ( ( yRef / ( 2 * refMbH ) ) * RefLayerPicWidthInMbs + ( xRef / refMbW ) ) \quad \text{(G-293)}$$

2. Depending on refLayerFieldMbFlag[ refMbAddrTop ], the variable refMbAddr is derived as follows.

   – If refLayerFieldMbFlag[ refMbAddrTop ] is equal to 0, the variable refMbAddr is derived by

$$refMbAddr = refMbAddrTop + ( yRef \% ( 2 * refMbH ) ) / refMbH \quad \text{(G-294)}$$

– Otherwise (refLayerFieldMbFlag[ refMbAddrTop ] is equal to 1), the variable refMbAddr is derived by

$$refMbAddr = refMbAddrTop + ( yRef \% 2 ) \quad \text{(G-295)}$$

The reference layer slice identification refSliceIdc is set equal to refLayerSliceIdc[ refMbAddr ].

Depending on refLayerMbType[ refMbAddr ], the variable refIntraMbFlag is derived as follows.

– If refLayerMbType[ refMbAddr ] is equal to I_4x4, I_8x8, I_16x16, I_PCM, or I_BL, refIntraMbFlag is set equal to 1.

– Otherwise (refLayerMbType[ refMbAddr ] is not equal to I_4x4, I_8x8, I_16x16, I_PCM, or I_BL), refIntraMbFlag is set equal to 0.

#### G.8.6.2.2.2 Construction process for not available sample values prior to intra resampling

Inputs to this process are

– two variables refMbW and refMbH specifying the width and height, respectively, of a reference layer macroblock for the considered colour component,

– two variables refArrayW and refArrayH specifying the width and height, respectively, of the array of reference layer sample values,

– a (refArrayW)x(refArrayH) array refSampleArray of reference layer sample values,

– two variables xOffset and yOffset specifying the x and y coordinates, respectively, of the reference layer sample location that corresponds to the sample refSampleArray[ 0, 0 ] of the array refSampleArray.

Output of this process is a modified version of the array refSampleArray.

For each sample refSampleArray[ x, y ] with x = (refMbW / 2)..(refArrayW – refMbW / 2 – 1) and y = (refMbH / 2)..(refArrayH – refMbH / 2 – 1) that is marked as "not available for Intra_Base prediction", the following ordered steps are specified.

1. The sample location difference ( xD, yD ) and the variable yA are derived by

$$xR = ( x + xOffset ) \% refMbW \quad \text{(G-296)}$$
$$yR = ( y + yOffset ) \% refMbH \quad \text{(G-297)}$$

$$xD = ( ( xR >= refMbW / 2 ) \ ? \ ( xR – refMbW ) \ : \ ( xR + 1 ) ) \quad \text{(G-298)}$$
$$yD = ( ( yR >= refMbH / 2 ) \ ? \ ( yR – refMbH ) \ : \ ( yR + 1 ) ) \quad \text{(G-299)}$$

$$yA = yD – 9 * Sign( yD ) \quad \text{(G-300)}$$

2. When any of the following conditions is true, yD is set equal to yA.
   – the sample refSampleArray[ x, y – yD ] is marked as "not available for Intra_Base prediction" and the sample refSampleArray[ x, y - yA ] is marked as "available for Intra_Base prediction"
   – all of the samples refSampleArray[ x – xD, y ], refSampleArray[ x, y – yD ], and refSampleArray[ x - xD, y – yD ] are marked as "not available for Intra_Base prediction"
3. The sample value refSampleArray[ x, y ] is derived as follows.
   – If the sample refSampleArray[ x – xD, y ] and the sample refSampleArray[ x, y – yD ] are marked as "available for Intra_Base prediction", the following ordered steps are specified.
     a. A variable cornerSampleAvailableFlag is derived as follows.
        – If the sample refSampleArray[ x – xD, y – yD ] is marked as "available for Intra_Base prediction", the variable cornerSampleAvailableFlag is set equal to 1.
        – Otherwise (the sample refSampleArray[ x – xD, y – yD ] is marked as "not available for Intra_Base prediction"), the variable cornerSampleAvailable is set equal to 0.
     b. The diagonal construction process for not available sample values as specified in subclause G.8.6.2.2.2.1 is invoked with refArrayW, refArrayH, refSampleArray, the sample location difference ( xD, yD ), the sample location ( x, y ), and the variable cornerSampleAvailableFlag as the inputs and the output is the sample array refSampleArray with a modified sample value at sample location ( x, y ).

– Otherwise (the sample refSampleArray[ x – xD, y ] or the sample refSampleArray[ x, y – yD ] is marked as "not available for Intra_Base prediction"), the following applies.

  – If the sample refSampleArray[ x – xD, y ] is marked as "available for Intra_Base prediction", the sample value refSampleArray[ x, y ] is set equal to refSampleArray[ x – xD, y ].

  – Otherwise, if the sample refSampleArray[ x, y – yD ] is marked as "available for Intra_Base prediction", the sample value refSampleArray[ x, y ] is set equal to refSampleArray[ x, y – yD ].

  – Otherwise (the sample refSampleArray[ x – xD, y ] and the sample refSampleArray[ x, y – yD ] are marked as "not available for Intra_Base prediction"), the sample value refSampleArray[ x, y ] is set equal to refSampleArray[ x – xD, y – yD ].

All samples refSampleArray[ x, y ] with x = 0..(refArrayW – 1) and y = 0..(refArrayH – 1) are marked as "available for Intra_Base prediction".

#### G.8.6.2.2.2.1 Diagonal construction process for not available sample values

Inputs to this process are

– two variables refArrayW and refArrayH specifying the width and height, respectively, of the array of reference layer sample values,

– a (refArrayW)x(refArrayH) array p of reference layer sample values,

– a sample location difference ( xD, yD ),

– a sample location ( x, y ) inside the reference layer sample array refSampleArray,

– a variable cornerSampleAvailableFlag.

Output of this process is the sample array p with a modified sample value at sample location ( x, y ).

The variables diffHorVer and sgnXY are derived by

$$diffHorVer = Abs( xD ) – Abs( yD ) \quad \text{(G-301)}$$
$$sgnXY = Sign( xD * yD ) \quad \text{(G-302)}$$

When cornerSampleAvailableFlag is equal to 1, the following applies.

1. The variable cornerSample is set equal to p[ x – xD, y – yD ].
2. The sample value p[ x – xD, y – yD ] is modified by

$$p[ x – xD, y – yD ] = ( p[ x – xD, y ] + p[ x, y – yD ] + 1 ) >> 1 \quad \text{(G-324a)}$$

The sample value p[ x, y ] is derived as follows.

– If diffHorVer is greater then 0, the sample location ( xC, yC ) is set equal to ( x – sgnXY * yD, y – yD ) and the sample value p[ x, y ] is derived by

$$p[ x, y ] = ( p[ xC – 1, yC ] + 2 * p[ xC, yC ] + p[ xC + 1, yC ] + 2 ) >> 2 \quad \text{(G-303)}$$

– Otherwise, if diffHorVer is less then 0, the sample location ( xC, yC ) is set equal to ( x – xD, y – sgnXY * xD ) and the sample value p[ x, y ] is derived by

$$p[ x, y ] = ( p[ xC, yC – 1 ] + 2 * p[ xC, yC ] + p[ xC, yC + 1 ] + 2 ) >> 2 \quad \text{(G-304)}$$

– Otherwise (diffVerHor is equal to 0), the sample value p[ x, y ] is derived by

$$p[ x, y ] = ( p[ x, y – yD ] + 2 * p[ x – xD, y – yD ] + p[ x – xD, y ] + 2 ) >> 2 \quad \text{(G-305)}$$

When cornerSampleAvailableFlag is equal to 1, the sample value p[ x – xD, y – yD ] is set equal to cornerSample.

#### G.8.6.2.3 Interpolation process for Intra_Base prediction

Inputs to this process are

– a variable filteringModeFlag specifying the interpolation method,

– a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,

– two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable botFieldFlag specifying whether a top or a bottom field is subject to the resampling process (when RefLayerFrameMbsOnlyFlag is equal to 0 or frame_mbs_only_flag is equal to 0),

– a variable fldPrdInFrmMbFlag specifying whether field prediction for a frame macroblock is applied,

– a variable yBorder specifying the vertical border for the output sample array predSamples,

– two variables refArrayW and refArrayH specifying the width and height, respectively, of the array of reference layer sample values,

– a (refArrayW)x(refArrayH) array refSampleArray of reference layer sample values,

– two variables xOffset and yOffset specifying the x and y coordinate, respectively, of the reference layer sample location that corresponds to the sample refSampleArray[ 0, 0 ] of the array refSampleArray.

Output of this process is an (mbW)x(mbH / ( 1 + fldPrdInFrmMbFlag ) + 2 * yBorder) array predArray of interpolated sample values.

Table G-8 specifies the filter coefficients eF[ p, x ] with p = 0..15 and x = 0..3 of the luma interpolation filter eF for resampling in Intra_Base prediction.

**Table G-8 – 16-phase luma interpolation filter for resampling in Intra_Base prediction**

| Phase p | Interpolation filter coefficients | | | |
|---|---|---|---|---|
| | eF[ p, 0 ] | eF[ p, 1 ] | eF[ p, 2 ] | eF[ p, 3 ] |
| 0 | 0 | 32 | 0 | 0 |
| 1 | -1 | 32 | 2 | -1 |
| 2 | -2 | 31 | 4 | -1 |
| 3 | -3 | 30 | 6 | -1 |
| 4 | -3 | 28 | 8 | -1 |
| 5 | -4 | 26 | 11 | -1 |
| 6 | -4 | 24 | 14 | -2 |
| 7 | -3 | 22 | 16 | -3 |
| 8 | -3 | 19 | 19 | -3 |
| 9 | -3 | 16 | 22 | -3 |
| 10 | -2 | 14 | 24 | -4 |
| 11 | -1 | 11 | 26 | -4 |
| 12 | -1 | 8 | 28 | -3 |
| 13 | -1 | 6 | 30 | -3 |
| 14 | -1 | 4 | 31 | -2 |
| 15 | -1 | 2 | 32 | -1 |

Let tempArray be a (refArrayW)x(mbH / ( 1 + fldPrdInFrmMbFlag ) + 2 * yBorder) array of samples. Each sample tempArray[ x, y ] with x = 0..(refArrayW – 1) and y = 0..(mbH / ( 1 + fldPrdInFrmMbFlag ) + 2 * yBorder – 1) is derived as specified in the following ordered steps.

1. The variable yP is derived by

$$yP = y * ( 1 + fldPrdInFrmMbFlag ) - yBorder + botFieldFlag \quad (G\text{-}306)$$

2. The derivation process for reference layer sample locations in resampling as specified in subclause G.6.3 is invoked with chromaFlag, the sample location ( 0, yP ), fieldMbFlag, and botFieldFlag as the inputs and the output is the sample location ( xRef16, yRef16 ) in units of 1/16-th sample.

   NOTE 1 – In this invocation of the process in subclause G.6.3, only the vertical component yRef16 of the sample location needs to be derived.

3. The variables yRef and yPhase are derived by

$$yRef = ( yRef16 >> 4 ) - yOffset \quad (G\text{-}307)$$
$$yPhase = ( yRef16 + 16 ) \% 16 \quad (G\text{-}308)$$

4. Depending on filteringModeFlag, the sample value tempArray[ x, y ] is derived as follows.

   – If filteringModeFlag is equal to 0, the sample value tempArray[ x, y ] is derived by

$$\begin{aligned} \text{tempArray}[\,x, y\,] = {} & \text{eF}[\,\text{yPhase}, 0\,] * \text{refSampleArray}[\,x, \text{yRef} - 1\,] + \\ & \text{eF}[\,\text{yPhase}, 1\,] * \text{refSampleArray}[\,x, \text{yRef}\,] + \\ & \text{eF}[\,\text{yPhase}, 2\,] * \text{refSampleArray}[\,x, \text{yRef} + 1\,] + \\ & \text{eF}[\,\text{yPhase}, 3\,] * \text{refSampleArray}[\,x, \text{yRef} + 2\,] \end{aligned} \tag{G-309}$$

   – Otherwise (filteringModeFlag is equal to 1), the sample value tempArray[ x, y ] is derived by

$$\begin{aligned} \text{tempArray}[\,x, y\,] = {} & (\,16 - \text{yPhase}\,) * \text{refSampleArray}[\,x, \text{yRef}\,] + \\ & \text{yPhase} * \text{refSampleArray}[\,x, \text{yRef} + 1\,] \end{aligned} \tag{G-310}$$

Each sample predArray[ x, y ] with x = 0..(mbW – 1) and y = 0..(mbH / ( 1 + fldPrdInFrmMbFlag ) + 2 * yBorder – 1) is derived as specified in the following ordered steps.

1. The derivation process for reference layer sample locations in resampling as specified in subclause G.6.3 is invoked with chromaFlag, the sample location ( x, 0 ), fieldMbFlag, and botFieldFlag as the inputs and the output is the sample location ( xRef16, yRef16 ) in units of 1/16-th sample.

   NOTE 2 – In this invocation of the process in subclause G.6.3, only the horizontal component xRef16 of the sample location needs to be derived.

2. The variables xRef and xPhase are derived by

$$\text{xRef} = (\,\text{xRef16} \gg 4\,) - \text{xOffset} \tag{G-311}$$

$$\text{xPhase} = (\,\text{xRef16} + 16\,) \,\%\, 16 \tag{G-312}$$

3. Depending on filteringModeFlag, and with Clip1 being replaced by $\text{Clip1}_\text{Y}$ for chromaFlag equal to 0 and $\text{Clip1}_\text{C}$ for chromaFlag equal to 1, the sample value predArray[ x, y ] is derived as follows.

   – If filteringModeFlag is equal to 0, the sample value tempArray[ x, y ] is derived by

$$\begin{aligned} \text{predArray}[\,x, y\,] = \text{Clip1}( {} & (\,\text{eF}[\,\text{xPhase}, 0\,] * \text{tempArray}[\,\text{xRef} - 1, y\,] + \\ & \text{eF}[\,\text{xPhase}, 1\,] * \text{tempArray}[\,\text{xRef}, y\,] + \\ & \text{eF}[\,\text{xPhase}, 2\,] * \text{tempArray}[\,\text{xRef} + 1, y\,] + \\ & \text{eF}[\,\text{xPhase}, 3\,] * \text{tempArray}[\,\text{xRef} + 2, y\,] + 512\,) \gg 10\,) \end{aligned} \tag{G-313}$$

   – Otherwise (filteringModeFlag is equal to 1), the sample value tempArray[ x, y ] is derived by

$$\begin{aligned} \text{predArray}[\,x, y\,] = {} & (\,(\,16 - \text{xPhase}\,) * \text{tempArray}[\,\text{xRef}, y\,] + \\ & \text{xPhase} * \text{tempArray}[\,\text{xRef} + 1, y\,] + 128\,) \gg 8 \end{aligned} \tag{G-314}$$

#### G.8.6.2.4 Vertical interpolation process for Intra_Base prediction

Inputs to this process are

– a variable filteringModeFlag specifying the interpolation method,

– a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,

– two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,

– a variable botFieldFlag specifying whether the sample array fieldPredArray contains interpolated samples for the top or bottom field,

– a variable yBorder specifying the vertical border for the sample array fieldPredArray,

– an (mbW)x(mbH / 2 + 2 * yBorder) array fieldPredArray of sample values.

Output of this process is an (mbW)x(mbH) array predArray of interpolated sample values.

Each sample predArray[ x, y ] with x = 0..(mbW – 1) and y = 0..(mbH – 1) is derived as follows.

– If ( y % 2 ) is equal to botFieldFlag, the sample value predArray[ x, y ] is derived by

$$\text{predArray}[\,x, y\,] = \text{fieldPredArray}[\,x, (\,y \gg 1\,) + \text{yBorder}\,] \tag{G-315}$$

– Otherwise (( y % 2 ) is not equal to botFieldFlag), the following ordered steps are specified.

1. The variable yFld is derived by

$$yFld = ( y >> 1 ) + yBorder - botFieldFlag \quad \text{(G-316)}$$

2. Depending on filteringModeFlag, and with Clip1 being replaced by $Clip1_Y$ for chromaFlag equal to 0 and $Clip1_C$ for chromaFlag equal to 1, the sample value predArray[ x, y ] is derived as follows.

   – If filteringModeFlag is equal to 0, the sample value predArray[ x, y ] is derived by

$$\begin{aligned} predArray[ x, y ] = Clip1( ( 19 * ( & fieldPredArray[ x, yFld ] + \\ & fieldPredArray[ x, yFld + 1 ] ) - \\ 3 * ( & fieldPredArray[ x, yFld - 1 ] + \\ & fieldPredArray[ x, yFld + 2 ] ) + 16 ) >> 5 ) \end{aligned} \quad \text{(G-317)}$$

   – Otherwise (filteringModeFlag is equal to 1), the sample value predArray[ x, y ] is derived by

$$\begin{aligned} predArray[ x, y ] = ( & fieldPredArray[ x, yFld ] + \\ & fieldPredArray[ x, yFld + 1 ] + 1 ) >> 1 \end{aligned} \quad \text{(G-318)}$$

### G.8.6.3 Resampling process for residual samples

Inputs to this process are

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerCTrafo with RefLayerPicSizeInMbs elements specifying the luma transform types for the macroblocks of the reference layer representation,

– a ($RefLayerPicWidthInSamples_C$)x($RefLayerPicHeightInSamples_C$) array $refLayerPicSamples_L$ of luma samples for the reference layer representation,

– a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array $picSamples_L$ of luma samples,

– when ChromaArrayType is not equal to 0, two ($RefLayerPicWidthInSamples_C$)x($RefLayerPicHeightInSamples_C$) arrays $refLayerPicSamples_{Cb}$ and $refLayerPicSamples_{Cr}$ of chroma samples for the reference layer representation,

– when ChromaArrayType is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ of chroma samples.

Outputs of this process are

– a modified version of the array $picSamples_L$ of luma samples,

– when ChromaArrayType is not equal to 0, modified versions of the arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ of chroma samples.

The resampling process for residual samples of a macroblock colour component as specified in subclause G.8.6.3.1 is invoked with chromaFlag equal to 0, mbW equal to 16, mbH equal to 16, fieldMbFlag, $refLayerPicSamples_L$, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and the output is the 16x16 array $mbPred_L$ of residual prediction samples for the luma component of the current macroblock.

When ChromaArrayType is not equal to 0, for CX being replaced by Cb and Cr, the resampling process for residual samples of a macroblock colour component as specified in subclause G.8.6.3.1 is invoked with chromaFlag equal to 1, mbW equal to MbWidthC, mbH equal to MbHeightC, fieldMbFlag, $refLayerPicSamples_{CX}$, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and the output is the (MbWidthC)x(MbHeightC) array $mbPred_{CX}$ of residual prediction samples for the CX component of the current macroblock.

The picture sample array construction process as specified in subclause G.8.5.4.1 is invoked with fieldMbFlag, $mbPred_L$, $picSamples_L$ and, when ChromaArrayType is not equal to 0, $mbPred_{Cb}$, $mbPred_{Cr}$, $picSamples_{Cb}$, and $picSamples_{Cr}$ as the inputs and the outputs are a modified version of $picSamples_L$ and, when ChromaArrayType is not equal to 0, modified versions of $picSamples_{Cb}$, and $picSamples_{Cr}$.

#### G.8.6.3.1 Resampling process for residual samples of a macroblock colour component

Inputs to this process are

- a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,
- two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,
- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- an array refLayerPicSamples, which is a (RefLayerPicWidthInSamples$_L$)x(RefLayerPicHeightInSamples$_L$) array containing constructed residual luma sample values for the reference layer representation when chromaFlag is equal to 0 or a (RefLayerPicWidthInSamples$_C$)x(RefLayerPicHeightInSamples$_C$) array containing constructed residual chroma sample values for the reference layer representation when chromaFlag is equal to 1,
- a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,
- a one-dimensional array refLayerCTrafo with RefLayerPicSizeInMbs elements specifying the luma transform types for the macroblocks of the reference layer representation.

Output of this process is an (mbW)x(mbH) array mbPred of residual prediction samples.

The variable botFieldFlag is derived as follows.

- If RefLayerFrameMbsOnlyFlag is equal to 0, botFieldFlag is set equal to 0.
- Otherwise, if field_pic_flag is equal to 1, botFieldFlag is set equal to bottom_field_flag.
- Otherwise, if RefLayerFieldPicFlag is equal to 1, botFieldFlag is set equal to RefLayerBottomFieldFlag.
- Otherwise, if fieldMbFlag is equal to 1, botFieldFlag is set equal to (CurrMbAddr % 2).
- Otherwise, botFieldFlag is set equal to 0.

The variable frameBasedResamplingFlag is derived as follows.

- If all of the following conditions are true, frameBasedResamplingFlag is set equal to 1.
  - RefLayerFrameMbsOnlyFlag is equal to 1
  - frame_mbs_only_flag is equal to 1
- Otherwise, frameBasedResamplingFlag is set equal to 0.

The variable topAndBotResamplingFlag is derived as follows.

- If all of the following conditions are true, topAndBotResamplingFlag is set equal to 1.
  - RefLayerFrameMbsOnlyFlag is equal to 0
  - RefLayerFieldPicFlag is equal to 0
  - frame_mbs_only_flag is equal to 0
  - fieldMbFlag is equal to 0
- Otherwise, topAndBotResamplingFlag is set equal to 0.

The array predArray is derived as specified in the following.

- If frameBasedResamplingFlag is equal to 1 or fieldMbFlag is equal to 1, the following ordered steps are specified.
  1. The reference layer sample array construction process prior to residual resampling as specified in subclause G.8.6.3.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, yBorder equal to 0, refLayerPicSamples, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and the outputs are the variables refArrayW, refArrayH, the array refSampleArray of reference layer sample values, the array refTransBlkIdc of reference layer transform block identifications, and the variables xOffset and yOffset.
  2. The interpolation process for residual prediction as specified in subclause G.8.6.3.3 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, fldPrdInFrmMbFlag equal to 0, yBorder equal to 0, refArrayW, refArrayH, refSampleArray, refTransBlkIdc, xOffset, and yOffset as the inputs and the output is the (mbW)x(mbH) array mbPred of residual prediction samples.

– Otherwise, if topAndBotResamplingFlag is equal to 0, the following ordered steps are specified.

1. The reference layer sample array construction process prior to residual resampling as specified in subclause G.8.6.3.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, yBorder equal to 1, refLayerPicSamples, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and the outputs are the variables refArrayW, refArrayH, the array refSampleArray of reference layer sample values, the array refTransBlkIdc of reference layer transform block identifications, and the variables xOffset and yOffset.
2. The interpolation process for residual prediction as specified in subclause G.8.6.3.3 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag, fldPrdInFrmMbFlag equal to 1, yBorder equal to 1, refArrayW, refArrayH, refSampleArray, refTransBlkIdc, xOffset, and yOffset as the inputs and the output is the (mbW)x(mbH / 2 + 2) array fieldPredArray of field prediction samples.
3. The vertical interpolation process for residual prediction as specified in subclause G.8.6.3.4 is invoked with mbW, mbH, botFieldFlag, yBorder, and fieldPredArray as the inputs and the output is the (mbW)x(mbH) array mbPred of residual prediction samples.

– Otherwise (topAndBotResamplingFlag is equal to 1), the following ordered steps are specified.

1. The reference layer sample array construction process prior to residual resampling as specified in subclause G.8.6.3.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 0, yBorder equal to 0, refLayerPicSamples, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and the outputs are the variables refArrayTopW, refArrayTopH, the array refSampleArrayTop of reference layer sample values, the array refTransBlkIdcTop of reference layer transform block identifications, and the variables xOffsetTop and yOffsetTop.
2. The interpolation process for residual prediction as specified in subclause G.8.6.3.3 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 0, fldPrdInFrmMbFlag equal to 1, yBorder equal to 0, refArrayTopW, refArrayTopH, refSampleArrayTop, refTransBlkIdcTop, xOffsetTop, and yOffsetTop as the inputs and the output is the (mbW)x(mbH / 2) array topFieldPredArray of top field prediction samples.
3. The reference layer sample array construction process prior to residual resampling as specified in subclause G.8.6.3.2 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 1, yBorder equal to 0, refLayerPicSamples, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and the outputs are the variables refArrayBotW, refArrayBotH, the array refSampleArrayBot of reference layer sample values, the array refTransBlkIdcBot of reference layer transform block identifications, and the variables xOffsetBot and yOffsetBot.
4. The interpolation process for residual prediction as specified in subclause G.8.6.3.3 is invoked with chromaFlag, mbW, mbH, fieldMbFlag, botFieldFlag equal to 1, fldPrdInFrmMbFlag equal to 1, yBorder equal to 0, refArrayBotW, refArrayBotH, refSampleArrayBot, refTransBlkIdcBot, xOffsetBot, and yOffsetBot as the inputs and the output is the (mbW)x(mbH / 2) array botFieldPredArray of bottom field prediction samples.
5. Each sample predArray[ x, y ] with x = 0..(mbW – 1) and y = 0..(mbH – 1) of the array mbPred of residual prediction samples is derived by

$$\begin{aligned} mbPred[\,x, y\,] = (\,(\,y \,\%\, 2\,) == 0\ ?\ & topFieldPredArray[\,x, y >> 1\,] \\ :\ & botFieldPredArray[\,x, y >> 1\,]\,) \end{aligned} \tag{G-319}$$

**G.8.6.3.2 Reference layer sample array construction process prior to residual resampling**

Inputs to this process are

– a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,

– two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,

– a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,

– a variable botFieldFlag specifying whether a top or a bottom field is subject to the resampling process (when RefLayerFrameMbsOnlyFlag is equal to 0 or frame_mbs_only_flag is equal to 0),

– a variable yBorder specifying the vertical border for determining the vertical size of the output arrays,

– an array refLayerPicSamples, which is a (RefLayerPicWidthInSamples$_L$)x(RefLayerPicHeightInSamples$_L$) array containing constructed residual luma sample values for the reference layer representation when chromaFlag is equal to 0 or a (RefLayerPicWidthInSamples$_C$)x(RefLayerPicHeightInSamples$_C$) array containing constructed residual chroma sample values for the reference layer representation when chromaFlag is equal to 1,

– a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,

– a one-dimensional array refLayerCTrafo with RefLayerPicSizeInMbs elements specifying the luma transform types for the macroblocks of the reference layer representation.

Outputs of this process are

– two variables refArrayW and refArrayH specifying the width and height, respectively, of the constructed arrays of reference layer sample values and reference layer transform block identification,

– a (refArrayW)x(refArrayH) array refSampleArray of reference layer sample values,

– a (refArrayW)x(refArrayH) array refTransBlkIdc of reference layer transform block identifications,

– two variables xOffset and yOffset specifying the x and y coordinate, respectively, of the reference layer sample location that corresponds to the sample refSampleArray[ 0, 0 ] of the array refSampleArray and the transform block identification refTransBlkIdc[ 0, 0 ] of the array refTransBlkIdc.

The variables refW, refH, refMbW, refMbH, xOffset, yOffset, refArrayW, refArrayH, yRefScale, and yRefAdd are derived as specified in the following ordered steps.

1. The derivation process for reference layer sample locations in resampling as specified in subclause G.6.3 is invoked with chromaFlag, the sample location ( 0, –yBorder ), fieldMbFlag, and botFieldFlag as the inputs and the output is the sample location ( xRefMin16, yRefMin16 ) in units of 1/16-th sample.

2. The derivation process for reference layer sample locations in resampling as specified in subclause G.6.3 is invoked with chromaFlag, the sample location ( mbW – 1, mbH – 1 + yBorder ), fieldMbFlag, and botFieldFlag as the inputs and the output is the sample location ( xRefMax16, yRefMax16 ) in units of 1/16-th sample.

3. With Z being replaced by L for chromaFlag equal to 0 and C for chromaFlag equal to 1, the variables refW, refH, refMbW, and refMbH are derived by

$$\begin{aligned}
refW &= RefLayerPicWidthInSamples_Z && \text{(G-320)}\\
refH &= RefLayerPicHeightInSamples_Z && \text{(G-321)}\\
refMbW &= ( ( chromaFlag == 0 ) \ ? \ 16 \ : \ RefLayerMbWidthC ) && \text{(G-322)}\\
refMbH &= ( ( chromaFlag == 0 ) \ ? \ 16 \ : \ RefLayerMbHeightC ) && \text{(G-323)}
\end{aligned}$$

4. The variables xOffset, yOffset, refArrayW, and refArrayH are derived by

$$\begin{aligned}
xOffset &= ( xRefMin16 >> 4 ) && \text{(G-324)}\\
yOffset &= ( yRefMin16 >> 4 ) && \text{(G-325)}\\
refArrayW &= ( ( xRefMax16 + 15 ) >> 4 ) - xOffset + 1 && \text{(G-326)}\\
refArrayH &= ( ( yRefMax16 + 15 ) >> 4 ) - yOffset + 1 && \text{(G-327)}
\end{aligned}$$

5. The variables yRefScale and yRefAdd are derived as follows.

   – If RefLayerFrameMbsOnlyFlag is equal to 1 or RefLayerFieldMbFlag is equal to 1, yRefScale is set equal to 1 and yRefAdd is set equal to 0.

   – Otherwise (RefLayerFrameMbsOnlyFlag is equal to 0 and RefLayerFieldMbFlag is equal to 0), yRefScale is set equal to 2 and yRefAdd is set equal to botFieldFlag.

Each sample refSampleArray[ x, y ] and each transform block identification refTransBlkIdc[ x, y ] with x = 0..(refArrayW – 1) and y = 0..(refArrayH – 1) are derived as specified in the following ordered steps.

1. A reference layer sample location ( xRef, yRef ) is derived by

$$\begin{aligned}
xRef &= max( 0, min( refW - 1, x + xOffset ) ) && \text{(G-328)}\\
yRef &= yRefScale * max( 0, min( refH / yRefScale - 1, y + yOffset ) ) + yRefAdd && \text{(G-329)}
\end{aligned}$$

2. The sample refSampleArray[ x, y ] is derived by

$$refSampleArray[ x, y ] = refLayerPicSamples[ xRef, yRef ] \qquad \text{(G-330)}$$

3. The transform block identification refTransBlkIdc[ x, y ] is derived by invoking the derivation process for reference layer transform block identifications as specified in subclause G.8.6.3.2.1 with the reference layer

sample location ( xRef, yRef ), chromaFlag, refMbW, refMbH, refLayerFieldMbFlag, and refLayerCTrafo as the inputs and assigning the output to refTransBlkIdc[ x, y ].

#### G.8.6.3.2.1 Derivation process for reference layer transform block identifications

Inputs to this process are

- a reference layer sample location ( xRef, yRef ) relative to the upper-left sample of the considered colour component of the reference layer picture,
- a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,
- two variables refMbW and refMbH specifying the width and height, respectively, of a reference layer macroblock for the considered colour component,
- a one-dimensional array refLayerFieldMbFlag with RefLayerPicSizeInMbs elements specifying which macroblocks of the reference layer representation are field macroblocks and which macroblocks are frame macroblocks,
- a one-dimensional array refLayerCTrafo with RefLayerPicSizeInMbs elements specifying the luma transform types for the macroblocks of the reference layer representation.

Output of this process is a variable refTransBlkIdc specifying an identification for the reference layer transform block that contains the sample at location ( xRef, yRef ).

The reference layer macroblock address refMbAddr and the reference layer sample location ( xM, yM ) inside the reference layer macroblock are derived as follows.

- If RefLayerMbaffFrameFlag is equal to 0, the variable refMbAddr and the sample location ( xM, yM ) are derived by

$$refMbAddr = ( yRef / refMbH ) * RefLayerPicWidthInMbs + ( xRef / refMbW ) \quad (G\text{-}331)$$
$$xM = xRef \,\%\, refMbW \quad (G\text{-}332)$$
$$yM = yRef \,\%\, refMbH \quad (G\text{-}333)$$

  Otherwise (RefLayerMbaffFrameFlag is equal to 1), the variable refMbAddr is derived as specified in the following ordered steps.

1. A variable refMbAddrTop and the horizontal sample location xM are derived by

$$refMbAddrTop = 2 * ( ( yRef / ( 2 * refMbH ) ) * RefLayerPicWidthInMbs + ( xRef / refMbW ) ) \quad (G\text{-}334)$$
$$xM = xRef \,\%\, refMbW \quad (G\text{-}335)$$

2. Depending on refLayerFieldMbFlag[ refMbAddrTop ], the variable refMbAddr and the vertical sample location yM are derived as follows.
   - If refLayerFieldMbFlag[ refMbAddrTop ] is equal to 0, the variables refMbAddr and yM are derived by

$$refMbAddr = refMbAddrTop + ( yRef \,\%\, ( 2 * refMbH) ) / refMbH \quad (G\text{-}336)$$
$$yM = yRef \,\%\, refMbH \quad (G\text{-}337)$$

   - Otherwise (refLayerFieldMbFlag[ refMbAddrTop ] is equal to 1), the variables refMbAddr and yM are derived by

$$refMbAddr = refMbAddrTop + ( yRef \,\%\, 2 ) \quad (G\text{-}338)$$
$$yM = ( yRef \,\%\, ( 2 * refMbH ) ) >> 1 \quad (G\text{-}339)$$

Depending on chromaFlag, RefLayerChromaArrayType, and refLayerCTrafo[ refMbAddr ], the following applies.

- If (chromaFlag is equal to 0 or RefLayerChromaArrayType is equal to 3) and refLayerCTrafo[ refMbAddr ] is equal to T_8x8, the variable refTransBlkIdc is derived by

$$refTransBlkIdc = 1 + 2 * ( 4 * refMbAddr + 2 * ( yM / 8 ) + ( xM / 8 ) ) \quad (G\text{-}340)$$

- Otherwise ((chromaFlag is equal to 1 and RefLayerChromaArrayType is not equal to 3) or refLayerCTrafo[ refMbAddr ] is not equal to T_8x8), the variable refTransBlkIdc is derived by

$$refTransBlkIdc = 2 * ( 16 * refMbAddr + 4 * ( yM / 4 ) + ( xM / 4 ) ) \quad (G\text{-}341)$$

#### G.8.6.3.3 Interpolation process for residual prediction

Inputs to this process are

- a variable chromaFlag specifying whether the luma or a chroma component is subject to the resampling process,
- two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,
- a variable fieldMbFlag specifying whether the current macroblock is a field or a frame macroblock,
- a variable botFieldFlag specifying whether a top or a bottom field is subject to the resampling process (when RefLayerFrameMbsOnlyFlag is equal to 0 or frame_mbs_only_flag is equal to 0),
- a variable fldPrdInFrmMbFlag specifying whether field prediction for a frame macroblock is applied,
- a variable yBorder specifying the vertical border for the output sample array predSamples,
- two variables refArrayW and refArrayH specifying the width and height, respectively, of the array of reference layer sample values and the array of transform block identifications,
- a (refArrayW)x(refArrayH) array refSampleArray of reference layer sample values,
- a (refArrayW)x(refArrayH) array refTransBlkIdc of transform block identifications,
- two variables xOffset and yOffset specifying the x and y coordinate, respectively, of the reference layer sample location that corresponds to the sample refSampleArray[ 0, 0 ] of the array refSampleArray and the transform block identification refTransBlkIdc[ 0, 0 ] of the array refTransBlkIdc.

Output of this process is an (mbW)x(mbH / ( 1 + fldPrdInFrmMbFlag ) + 2 * yBorder) array predArray of interpolated sample values.

Each sample predArray[ x, y ] with x = 0..(mbW – 1) and y = 0..(mbH / ( 1 + fldPrdInFrmMbFlag ) + 2 *yBorder – 1) is derived as specified in the following ordered steps.

1. The variable yP is derived by

$$yP = y * ( 1 + fldPrdInFrmMbFlag ) - yBorder + botFieldFlag \quad \text{(G-342)}$$

2. The derivation process for reference layer sample locations in resampling as specified in subclause G.6.3 is invoked with chromaFlag, the sample location ( x, yP ), fieldMbFlag, and botFieldFlag as the inputs and the output is the sample location ( xRef16, yRef16 ) in units of 1/16-th sample.

3. The variables xRef, yRef, xPhase, and yPhase are derived by

$$xRef = ( xRef16 >> 4 ) - xOffset \quad \text{(G-343)}$$
$$yRef = ( yRef16 >> 4 ) - yOffset \quad \text{(G-344)}$$
$$xPhase = ( xRef16 + 16 ) \% 16 \quad \text{(G-345)}$$
$$yPhase = ( yRef16 + 16 ) \% 16 \quad \text{(G-346)}$$

4. Let tempPred by a one-dimensional array with 2 elements. Each sample value tempPred[ dY ] with dY = 0..1 is derived as follows.

   - If refTransBlkIdc[ xRef, yRef + dY ] is equal to refTransBlkIdc[ xRef + 1, yRef + dY ], the sample value tempPred[ dY ] is derived by

$$tempPred[ dY ] = ( 16 - xPhase ) * refSampleArray[ xRef, yRef + dY ] + xPhase * refSampleArray[ xRef + 1, yRef + dY ] \quad \text{(G-347)}$$

   - Otherwise (refTransBlkIdc[ xRef, yRef + dY ] is not equal to refTransBlkIdc[ xRef + 1, yRef + dY ]), the sample value tempPred[ dY ] is derived by

$$tempPred[ dY ] = ( ( xPhase < 8 ) \ ? \ refSampleArray[ xRef, yRef + dY ] : refSampleArray[ xRef + 1, yRef + dY ] ) \quad \text{(G-348)}$$

5. The sample value predArray[ x, y ] is derived as follows.

   - If refTransBlkIdc[ xRef, yRef ] is equal to refTransBlkIdc[ xRef, yRef + 1 ], the sample value predArray[ x, y ] is derived by

$$predArray[ x, y ] = ( ( 16 - yPhase ) * tempPred[ 0 ] + yPhase * tempPred[ 1 ] + 128 ) >> 8 \quad \text{(G-349)}$$

– Otherwise (refTransBlkIdc[ xRef, yRef ] is not equal to refTransBlkIdc[ xRef, yRef + 1 ]), the sample value predArray[ x, y ] is derived by

$$predArray[ dY ] = ( ( ( yPhase < 8 ) \ ? \ tempPred[ 0 ] \ : \ tempPred[ 1 ] ) + 8 ) >> 4 \quad \text{(G-350)}$$

#### G.8.6.3.4 Vertical interpolation process for residual prediction

Inputs to this process are

– two variables mbW and mbH specifying the width and height, respectively, of a macroblock for the considered colour component,

– a variable botFieldFlag specifying whether the sample array fieldPredArray contains interpolated samples for the top or bottom field,

– a variable yBorder specifying the vertical border for the sample array fieldPredArray,

– an (mbW)x(mbH / 2 + 2 * yBorder) array fieldPredArray of sample values.

Output of this process is an (mbW)x(mbH) array predArray of interpolated sample values.

Each sample predArray[ x, y ] with x = 0..(mbW – 1) and y = 0..(mbH – 1) is derived as follows.

– If ( y % 2 ) is equal to botFieldFlag, the sample value predArray[ x, y ] is derived by

$$predArray[ x, y ] = fieldPredArray[ x, ( y >> 1 ) + yBorder ] \quad \text{(G-351)}$$

– Otherwise (( y % 2 ) is not equal to botFieldFlag), the sample value predArray[ x, y ] is derived by

$$predArray[ x, y ] = ( fieldPredArray[ x, ( y >> 1 ) + yBorder – botFieldFlag ] + fieldPredArray[ x, ( y >> 1 ) + yBorder – botFieldFlag + 1 ] + 1 ) >> 1 \quad \text{(G-352)}$$

### G.8.7 SVC deblocking filter processes

Subclause G.8.7.1 specifies the deblocking filter process for Intra_Base prediction

Subclause G.8.7.2 specifies the deblocking filter process for target slices

#### G.8.7.1 Deblocking filter process for Intra_Base prediction

Inputs to the process are

– the variable dependencyIdRefLayer

– the collective term currentVars

Output of this process is a modified version of currentVars.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the layer representation with DQId equal to (dependencyIdRefLayer << 4).

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

The derivation process for quantisation parameters used in the deblocking filter process as specified in subclause G.8.7.3 is invoked with mbType, $tQP_Y$, and tCoeffLevel as the inputs and the outputs are a list $qpDB_Y$ specifying luma quantisation parameter that are used in the deblocking filter process and, when ChromaArrayType is not equal to 0, two lists $qpDB_{Cb}$ and $qpDB_{Cr}$ specifying chroma quantisation parameters that are used in the deblocking filter process.

Let disableDeblockingFilterIdc, filterOffsetA, and filterOffsetB be equal to the value of disable_inter_layer_deblocking_filter_idc, InterlayerFilterOffsetA, and InterlayerFilterOffsetB, respectively, for the layer representation with DQId equal to (dependency_id << 4).

For the current macroblock address CurrMbAddr proceeding over values 0..(PicSizeInMbs – 1), the following applies.

– The macroblock deblocking filter process as specified in subclause G.8.7.4 is invoked with interLayerDeblockingFlag = 1, disableDeblockingFilterIdc, filterOffsetA, filterOffsetB, sliceBoundariesOnlyFlag = 0, currentVars, $qpDB_Y$ and, when ChromaArrayType is not equal to 0, $qpDB_{Cb}$ and $qpDB_{Cr}$ as the inputs and the output is a modified version of currentVars.

When disable_deblocking_filter_idc is equal to 3 or 6, for the current macroblock address CurrMbAddr proceeding over values 0..(PicSizeInMbs – 1), the following applies.

– The macroblock deblocking filter process as specified in subclause G.8.7.4 is invoked with interLayerDeblockingFlag = 1, disableDeblockingFilterIdc, filterOffsetA, filterOffsetB, sliceBoundariesOnlyFlag = 1, currentVars, $qpDB_Y$ and, when ChromaArrayType is not equal to 0, $qpDB_{Cb}$ and $qpDB_{Cr}$ as the inputs and the output is a modified version of currentVars.

### G.8.7.2 Deblocking filter process for target representations

Input to the process is the collective term currentVars

Output of this process are decoded samples represented by the sample arrays $S_L$, and, when ChromaArrayType is not equal to 0, $S_{Cb}$ and $S_{Cr}$.

Unless stated otherwise, the syntax elements and derived upper-case variables that are referred to by the process specified in this subclause and all child processes invoked from this process are the syntax elements and derived upper-case variables for the layer representation with DQId equal to (dependencyId << 4).

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

The derivation process for quantisation parameters used in the deblocking filter process as specified in subclause G.8.7.3 is invoked with mbType, $tQP_Y$, and tCoeffLevel as the inputs and the outputs are a list $qpDB_Y$ specifying luma quantisation parameter that are used in the deblocking filter process and, when ChromaArrayType is not equal to 0, two lists $qpDB_{Cb}$ and $qpDB_{Cr}$ specifying chroma quantisation parameters that are used in the deblocking filter process.

For the current macroblock address CurrMbAddr proceeding over values 0..(PicSizeInMbs – 1), the following applies.

– Let disableDeblockingFilterIdc, filterOffsetA, and filterOffsetB be equal to the value of disable_deblocking_filter_idc, FilterOffsetA, and FilterOffsetB, respectively, for the slice with DQId equal to (sliceIdc[ CurrMbAddr ] & 127) and first_mb_in_slice equal to (sliceIdc[ CurrMbAddr ] >> 7).

– The macroblock deblocking filter process as specified in subclause G.8.7.4 is invoked with interLayerDeblockingFlag = 0, disableDeblockingFilterIdc, filterOffsetA, filterOffsetB, sliceBoundariesOnlyFlag = 0, currentVars, $qpDB_Y$ and, when ChromaArrayType is not equal to 0, $qpDB_{Cb}$ and $qpDB_{Cr}$ as the inputs and the output is a modified version of currentVars.

When disable_deblocking_filter_idc is equal to 3 or 6, for the current macroblock address CurrMbAddr proceeding over values 0..(PicSizeInMbs – 1), the following applies.

– Let disableDeblockingFilterIdc, filterOffsetA, and filterOffsetB be equal to the value of disable_deblocking_filter_idc, FilterOffsetA, and FilterOffsetB, respectively, for the slice with DQId equal to (sliceIdc[ CurrMbAddr ] & 127) and first_mb_in_slice equal to (sliceIdc[ CurrMbAddr ] >> 7).

– The macroblock deblocking filter process as specified in subclause G.8.7.4 is invoked with interLayerDeblockingFlag = 0, disableDeblockingFilterIdc, filterOffsetA, filterOffsetB, sliceBoundariesOnlyFlag = 1, currentVars, $qpDB_Y$ and, when ChromaArrayType is not equal to 0, $qpDB_{Cb}$ and $qpDB_{Cr}$ as inputs and the output is a modified version of currentVars.

### G.8.7.3 Derivation process for quantisation parameters used in the deblocking filter process

Inputs to this process are

– a one-dimensional array mbType with PicSizeInMbs elements specifying macroblock types for the macroblocks of the current decoded or partly decoded dependency representation,

– a one-dimensional array $tQP_Y$ with PicSizeInMbs elements specifying luma quantisation parameters for the macroblocks of the current decoded or partly decoded dependency representation,

– a (PicSizeInMbs)x(256 + 2 * MbWidthC * MbHeightC) array tCoeffLevel specifying transform coefficient level values for the macroblocks of the current decoded or partly decoded dependency representation.

Outputs of this process are

– a one-dimensional array $qpDB_Y$ with PicSizeInMbs elements specifying luma quantisation parameters used in the deblocking filter process for the macroblocks of the current decoded or partly decoded dependency representation,

– when ChromaArrayType is not equal to 0, two one-dimensional arrays $qpDB_{Cb}$ and $qpDB_{Cr}$ with PicSizeInMbs elements specifying chroma quantisation parameters used in the deblocking filter process for the macroblocks of the current decoded or partly decoded dependency representation.

The macroblock address mbAddr proceeds over the values 0..(PicSizeInMbs – 1), and for each value of mbAddr, the following ordered steps are specified.

1. The variable $qpDB_Y$[ mbAddr ] is derived as follows.
   - If mbType[ mbAddr ] is equal to I_PCM, $qpDB_Y$[ mbAddr ] is set equal to 0.
   - Otherwise, if tcoeff_level_prediction_flag is equal to 1, mbAddr is greater than 0, mbType[ mbAddr ] is not equal to I_16x16, and all elements tCoeffLevel[ mbAddr ][ i ] with i = 0..(255 + 2 * MbWidthC * MbHeightC) are equal to 0, $qpDB_Y$[ mbAddr ] is set equal to $qpDB_Y$[ mbAddr – 1 ].
   - Otherwise, $qpDB_Y$[ mbAddr ] is set equal to $tQP_Y$[ mbAddr ].
2. When ChromaArrayType is not equal to 0, for C being replaced by Cb and Cr, the variable $qpDB_C$[ mbAddr ] is set equal to the value of $QP_C$ that corresponds to a value of $qpDB_Y$[ mbAddr ] for $QP_Y$ as specified in subclause 8.5.8. During this invocation of the process in subclause 8.5.8, the syntax elements chroma_qp_index_offset and second_chroma_qp_index_offset of the layer representation with DQId equal to DQIdMax are used.

### G.8.7.4 Macroblock deblocking filter process

Inputs to this process are

- the variables interLayerDeblockingFlag, disableDeblockingFilterIdc, filterOffsetA, filterOffsetB, and sliceBoundariesOnlyFlag,
- the collective term currentVars,
- a one-dimensional array $qpDB_Y$ with PicSizeInMbs elements specifying luma quantisation parameters used in the deblocking filter process for the macroblocks of the current decoded or partly decoded dependency representation,
- when ChromaArrayType is not equal to 0, two one-dimensional arrays $qpDB_{Cb}$ and $qpDB_{Cr}$ with PicSizeInMbs elements specifying chroma quantisation parameters used in the deblocking filter process for the macroblocks of the current decoded or partly decoded dependency representation.

Output of this process is a modified version of currentVars

In the following of this subclause, the variables sliceIdc, fieldMbFlag, baseModeFlag, mbType, lumaTrafo, subMbType, ipred4x4, ipred8x8, ipred16x16, ipredChroma, refIdxL0, refIdxL1, predFlagL0, predFlagL1. mvL0, mvL1, mvCnt, sTCoeff, tCoeffLevel, $tQP_Y$, $tQP_{Cb}$, $tQP_{Cr}$, $rS_L$, $rS_{Cb}$, $rS_{Cr}$, $cS_L$, $cS_{Cb}$, and $cS_{Cr}$ of the collective term currentVars are referred to as sliceIdc, fieldMbFlag, baseModeFlag, mbType, lumaTrafo, subMbType, ipred4x4, ipred8x8, ipred16x16, ipredChroma, refIdxL0, refIdxL1, predFlagL0, predFlagL1, mvL0, mvL1, mvCnt, sTCoeff, tCoeffLevel, $tQP_Y$, $tQP_{Cb}$, $tQP_{Cr}$, $rS_L$, $rS_{Cb}$, $rS_{Cr}$, $cS_L$, $cS_{Cb}$, and $cS_{Cr}$, respectively.

The derivation process for neighbouring macroblocks specified in subclause 6.4.10.1 is invoked and the output is assigned to mbAddrA and mbAddrB. For this invocation of the process in subclause 6.4.10.1, the current macroblock is treated as field macroblock when fieldMbFlag[ CurrMbAddr ] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[ CurrMbAddr ] is equal to 0.

NOTE – The availability status of the macroblocks mbAddrA and mbAddrB is not used inside this subclause. Slice boundaries are detected using the array sliceIdc.

The variable filterLeftLumaMbEdgeFlag is derived as follows.

- If any of the following conditions is true, the variable filterLeftLumaMbEdgeFlag is set equal to 0.
  - MbaffFrameFlag is equal to 0 and CurrMbAddr % PicWidthInMbs is equal to 0
  - MbaffFrameFlag is equal to 1 and ( CurrMbAddr >> 1 ) % PicWidthInMbs is equal to 0
  - disableDeblockingFilterIdc is equal to 1
  - disableDeblockingFilterIdc is equal to 2 or 5 and sliceIdc[ mbAddrA ] is different than sliceIdc[ CurrMbAddr ]
  - disableDeblockingFilterIdc is equal to 3 or 6, sliceBoundariesOnlyFlag is equal to 0, and sliceIdc[ mbAddrA ] is different than sliceIdc[ CurrMbAddr ]
  - disableDeblockingFilterIdc is equal to 3 or 6, sliceBoundariesOnlyFlag is equal to 1, and sliceIdc[ mbAddrA ] is equal to sliceIdc[ CurrMbAddr ]

- interLayerDeblockingFlag is equal to 1 and mbType[ CurrMbAddr ] specifies an Inter macroblock prediction mode
- interLayerDeblockingFlag is equal to 1 and mbType[ mbAddrA ] specifies an Inter macroblock prediction mode

– Otherwise, the variable filterLeftLumaMbEdgeFlag is set equal to 1.

The variable filterTopLumaMbEdgeFlag is derived as follows.

– If any of the following conditions is true, the variable filterTopLumaMbEdgeFlag is set equal to 0.
  - MbaffFrameFlag is equal to 0 and CurrMbAddr is less than PicWidthInMbs
  - MbaffFrameFlag is equal to 1, ( CurrMbAddr >> 1 ) is less than PicWidthInMbs, and fieldMbFlag[ CurrMbAddr ] is equal to 1
  - MbaffFrameFlag is equal to 1, ( CurrMbAddr >> 1 ) is less than PicWidthInMbs, fieldMbFlag[ CurrMbAddr ] is equal to 0, and CurrMbAddr % 2 is equal to 0
  - disableDeblockingFilterIdc is equal to 1
  - disableDeblockingFilterIdc is equal to 2 or 5 and sliceIdc[ mbAddrB ] is different than sliceIdc[ CurrMbAddr ]
  - disableDeblockingFilterIdc is equal to 3 or 6, sliceBoundariesOnlyFlag is equal to 0, and sliceIdc[ mbAddrB ] is different than sliceIdc[ CurrMbAddr ]
  - disableDeblockingFilterIdc is equal to 3 or 6, sliceBoundariesOnlyFlag is equal to 1, and sliceIdc[ mbAddrB ] is equal to sliceIdc[ CurrMbAddr ]
  - interLayerDeblockingFlag is equal to 1 and mbType[ CurrMbAddr ] specifies an Inter macroblock prediction mode
  - interLayerDeblockingFlag is equal to 1 and mbType[ mbAddrB ] specifies an Inter macroblock prediction mode
– Otherwise, the variable filterTopLumaMbEdgeFlag is set equal to 1.

The variable filterInternalLumaEdgesFlag is derived as follows.

– If any of the following conditions is true, the variable filterInternalLumaEdgesFlag is set equal to 0.
  - disableDeblockingFilterIdc is equal to 1
  - disableDeblockingFilterIdc is equal to 3 or 6 and sliceBoundariesOnlyFlag is equal to 1
  - interLayerDeblockingFlag is equal to 1 mbType[ CurrMbAddr ] specifies an Inter macroblock prediction mode
– Otherwise the variable filterInternalLumaEdgesFlag is set equal to 1.

The variables filterLeftChromaMbEdgeFlag, filterTopChromaMbEdgeFlag, and filterInternalChromaEdgesFlag are derived as follows.

– If disableDeblockingFilterIdc is greater than 3, filterLeftChromaMbEdgeFlag, filterTopChromaMbEdgeFlag, and filterInternalChromaEdgesFlag are set equal to 0.
– Otherwise (disableDeblockingFilterIdc is less than 4), filterLeftChromaMbEdgeFlag, filterTopChromaMbEdgeFlag, and filterInternalChromaEdgesFlag are set equal to filterLeftLumaMbEdgeFlag, filterTopLumaMbEdgeFlag, and filterInternalLumaEdgesFlag, respectively.

The variable fieldModeMbFlag is derived as follows.

– If any of the following conditions is true, fieldModeMbFlag is set equal to 1.
  - field_pic_flag is equal to 1
  - MbaffFrameFlag is equal to 1 and fieldMbFlag[ CurrMbAddr ] is equal to 1
– Otherwise, fieldModeMbFlag is set equal to 0.

When filterLeftLumaMbEdgeFlag is equal to 1, the filtering of the left vertical luma edge is specified as follows.

– The process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 1, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_Y$[ mbAddrA ], $qP_q$ set equal to $qpDB_Y$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (0, k) with k = 0..15, and $cS_L$ as inputs and $cS_L$ as output.

When filterInternalLumaEdgesFlag is equal to 1, the filtering of the internal vertical luma edges is specified as follows.

– When lumaTrafo[ CurrMbAddr ] is not equal to T_8x8, the process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 1, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_Y$[ CurrMbAddr ], $qP_q$ set equal to $qpDB_Y$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (4, k) with k = 0..15, and $cS_L$ as inputs and $cS_L$ as output.

– The process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 1, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_Y$[ CurrMbAddr ], $qP_q$ set equal to $qpDB_Y$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (8, k) with k = 0..15, and $cS_L$ as inputs and $cS_L$ as output.

– When lumaTrafo[ CurrMbAddr ] is not equal to T_8x8, the process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 1, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_Y$[ CurrMbAddr ], $qP_q$ set equal to $qpDB_Y$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (12, k) with k = 0..15, and $cS_L$ as inputs and $cS_L$ as output.

When filterTopLumaMbEdgeFlag is equal to 1, the filtering of the top horizontal luma edge is specified as follows.

– If MbaffFrameFlag is equal to 1, (CurrMbAddr % 2) is equal to 0, CurrMbAddr is greater than or equal to (2 * PicWidthInMbs), fieldMbFlag[ CurrMbAddr ] is equal to 0, and fieldMbFlag[ CurrMbAddr - 2 * PicWidthInMbs + 1 ] is equal to 1, the following applies.

  – The process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 0, fieldModeFilteringFlag set equal to 1, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_Y$[ CurrMbAddr - 2 * PicWidthInMbs ], $qP_q$ set equal to $qpDB_Y$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (k, 0) with k = 0..15, and $cS_L$ as inputs and $cS_L$ as output.

  – The process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 0, fieldModeFilteringFlag set equal to 1, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_Y$[ CurrMbAddr - 2 * PicWidthInMbs + 1 ], $qP_q$ set equal to $qpDB_Y$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (k, 1) with k = 0..15, and $cS_L$ as inputs and $cS_L$ as output.

– Otherwise, the process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 0, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_Y$[ mbAddrB ], $qP_q$ set equal to $qpDB_Y$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (k, 0) with k = 0..15, and $cS_L$ as inputs and $cS_L$ as output.

When filterInternalLumaEdgesFlag is equal to 1, the filtering of the internal horizontal luma edges is specified as follows.

– When lumaTrafo[ CurrMbAddr ] is not equal to T_8x8, the process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 0, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_Y$[ CurrMbAddr ], $qP_q$ set equal to $qpDB_Y$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (k, 4) with k = 0..15, and $cS_L$ as inputs and $cS_L$ as output.

– The process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 0, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_Y$[ CurrMbAddr ], $qP_q$ set equal to $qpDB_Y$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (k, 8) with k = 0..15, and $cS_L$ as inputs and $cS_L$ as output.

– When lumaTrafo[ CurrMbAddr ] is not equal to T_8x8, the process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 0, verticalEdgeFlag set equal to 0, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_Y$[ CurrMbAddr ], $qP_q$ set equal to $qpDB_Y$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (k, 12) with k = 0..15, and $cS_L$ as inputs and $cS_L$ as output.

When ChromaArrayType is not equal to 0, for the filtering of both chroma components with C being replaced by Cb and Cr in $qpDB_C$ and $cS_C$, the following applies.

– When filterLeftChromaMbEdgeFlag is equal to 1, the filtering of the left vertical chroma edge is specified as follows.

  – The process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 1, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_C$[ mbAddrA ], $qP_q$ set equal to $qpDB_C$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (0, k) with k = 0..(MbHeightC – 1), and $cS_C$ as inputs and $cS_C$ as output.

– When filterInternalChromaEdgesFlag is equal to 1, the filtering of the internal vertical chroma edge is specified as follows.

  – When ChromaArrayType is not equal to 3 or transform_size_8x8_flag is equal to 0, the process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 1, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_C$[ CurrMbAddr ], $qP_q$ set equal to $qpDB_C$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (4, k) with k = 0..(MbHeightC – 1), and $cS_C$ as inputs and $cS_C$ as output.

  – When ChromaArrayType is equal to 3, the process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 1, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_C$[ CurrMbAddr ], $qP_q$ set equal to $qpDB_C$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (8, k) with k = 0..(MbHeightC – 1), and $cS_C$ as inputs and $cS_C$ as output.

  – When ChromaArrayType is equal to 3, the process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 1, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_C$[ CurrMbAddr ], $qP_q$ set equal to $qpDB_C$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (12, k) with k = 0..(MbHeightC – 1), and $cS_C$ as inputs and $cS_C$ as output.

– When filterTopChromaMbEdgeFlag is equal to 1, the filtering of the top horizontal chroma edge is specified as follows.

  – If MbaffFrameFlag is equal to 1, (CurrMbAddr % 2) is equal to 0, CurrMbAddr is greater than or equal to (2 * PicWidthInMbs), fieldMbFlag[ CurrMbAddr ] is equal to 0, fieldMbFlag[ CurrMbAddr - 2 * PicWidthInMbs + 1 ] is equal to 1, the following applies.

    – The process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 0, fieldModeFilteringFlag set equal to 1, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_C$[ CurrMbAddr – 2 * PicWidthInMbs ], $qP_q$ set equal to $qpDB_C$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (k, 0) with k = 0..(MbWidthC – 1), and $cS_C$ as inputs and $cS_C$ as output.

    – The process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 0, fieldModeFilteringFlag set equal to 1, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_C$[ CurrMbAddr – 2 * PicWidthInMbs + 1 ], $qP_q$ set equal to $qpDB_C$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (k, 1) with k = 0..(MbWidthC – 1), and $cS_C$ as inputs and $cS_C$ as output.

  – Otherwise, the process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 0, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_C$[ mbAddrB ], $qP_q$ set equal to $qpDB_C$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (k, 0) with k = 0..(MbWidthC – 1), and $cS_C$ as inputs and $cS_C$ as output.

– When filterInternalChromaEdgesFlag is equal to 1, the filtering of the internal horizontal chroma edge is specified as follows.

  – When ChromaArrayType is not equal to 3 or transform_size_8x8_flag is equal to 0, the process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 0, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_C$[ CurrMbAddr ], $qP_q$ set equal to $qpDB_C$[ CurrMbAddr ], currentVars, $(xE_k, yE_k)$ set equal to (k, 4) with k = 0..(MbWidthC – 1), and $cS_C$ as inputs and $cS_C$ as output.

  – When ChromaArrayType is not equal to 1, the process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 0, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_C$[ CurrMbAddr ], $qP_q$ set equal to

$qpDB_C$[ CurrMbAddr ], currentVars, ($xE_k$, $yE_k$) set equal to (k, 8) with k = 0..(MbWidthC – 1), and $cS_C$ as inputs and $cS_C$ as output.

– When ChromaArrayType is equal to 2, the process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 0, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_C$[ CurrMbAddr ], $qP_q$ set equal to $qpDB_C$[ CurrMbAddr ], currentVars, ($xE_k$, $yE_k$) set equal to (k, 12) with k = 0..(MbWidthC – 1), and $cS_C$ as inputs and $cS_C$ as output.

– When ChromaArrayType is equal to 3 or transform_size_8x8_flag is equal to 0, the process specified in subclause G.8.7.4.1 is invoked with chromaEdgeFlag set equal to 1, verticalEdgeFlag set equal to 0, fieldModeFilteringFlag set equal to fieldModeMbFlag, filterOffsetA, filterOffsetB, $qP_p$ set equal to $qpDB_C$[ CurrMbAddr ], $qP_q$ set equal to $qpDB_C$[ CurrMbAddr ], currentVars, ($xE_k$, $yE_k$) set equal to (k, 12) with k = 0..(MbWidthC – 1), and $cS_C$ as inputs and $cS_C$ as output.

NOTE 1 – When field mode filtering (fieldModeFilteringFlag is equal to 1) is applied across the top horizontal edges of a frame macroblock, this vertical filtering across the top or bottom macroblock boundary may involve some samples that extend across an internal block edge that is also filtered internally in frame mode.

NOTE 2 – For example, in 4:2:0 chroma format when transform_size_8x8_flag is equal to 0, the following applies. 3 horizontal luma edges, 1 horizontal chroma edge for Cb, and 1 horizontal chroma edge for Cr are filtered that are internal to a macroblock. When field mode filtering (fieldModeFilteringFlag is equal to 1) is applied to the top edges of a frame macroblock, 2 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr between the frame macroblock and the above macroblock pair are filtered using field mode filtering, for a total of up to 5 horizontal luma edges, 3 horizontal chroma edges for Cb, and 3 horizontal chroma edges for Cr filtered that are considered to be controlled by the frame macroblock. In all other cases, at most 4 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr are filtered that are considered to be controlled by a particular macroblock.

**G.8.7.4.1 SVC filtering process for block edges**

Inputs to this process are

– the variable chromaEdgeFlag,

– the variable verticalEdgeFlag,

– the variable fieldModeFilteringFlag,

– the variables filterOffsetA and filterOffsetB,

– the variables $qP_p$ and $qP_q$,

– the collective term currentVars,

– a set of nE sample locations ($xE_k$, $yE_k$), with k = 0..(nE – 1), expressed relative to the upper left corner of the macroblock CurrMbAddr. The set of sample locations ($xE_k$, $yE_k$) represent the sample locations immediately to the right of a vertical edge (when verticalEdgeFlag is equal to 1) or immediately below a horizontal edge (when verticalEdgeFlag is equal to 0),

– an array of samples s'.

Output of this process is a modified version of the array s'.

The variable nE is derived as follows

– If chromaEdgeFlag is equal to 0, nE is set equal to 16.

– Otherwise (chromaEdgeFlag is equal to 1), nE is set equal to ( ( verticalEdgeFlag == 1 ) ? MbHeightC : MbWidthC ).

Inside this subclause, the arrays that are collectively referred to as currentVars are referred to by their names as specified in subclause G.8.1.2.1.

The variable dy is derived as follows.

– If fieldModeFilteringFlag is equal to 1 and MbaffFrameFlag is equal to 1, dy is set equal to 2.

– Otherwise (fieldModeFilteringFlag is equal to 0 or MbaffFrameFlag is equal to 0), dy is set equal to 1.

The position of the upper-left luma sample of the macroblock CurrMbAddr is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddr = CurrMbAddr as input and the output being assigned to ( xI, yI ). During the process in subclause 6.4.1, the current macroblock is treated as field macroblock when

fieldMbFlag[ CurrMbAddr ] is equal to 1, and it is treated as frame macroblock when fieldMbFlag[ CurrMbAddr ] is equal to 0.

The variables xP and yP are derived as follows.

– If chromaEdgeFlag is equal to 0, xP is set equal to xI and yP is set equal to yI.

– Otherwise (chromaEdgeFlag is equal to 1), xP is set equal to (xI / SubWidthC) and yP is set equal to ( ( yI + SubHeightC – 1 ) / SubHeightC ).

For each sample location ( $xE_k$, $yE_k$ ), k = 0..(nE – 1), the following applies.

– The filtering process is applied to a set of eight samples across a 4x4 block horizontal or vertical edge denoted as $p_i$ and $q_i$ with i = 0..3 as shown in Figure 8-11 with the edge lying between $p_0$ and $q_0$. $p_i$ and $q_i$ with i = 0..3 are specified as follows.

  – If verticalEdgeFlag is equal to 1,

$$q_i = s'[\ xP + xE_k + i,\ yP + dy * yE_k\ ] \quad \text{(G-353)}$$
$$p_i = s'[\ xP + xE_k – i – 1,\ yP + dy * yE_k\ ] \quad \text{(G-354)}$$

  – Otherwise (verticalEdgeFlag is equal to 0),

$$q_i = s'[\ xP + xE_k,\ yP + dy * ( yE_k + i ) – (yE_k \% 2 )\ ] \quad \text{(G-355)}$$
$$p_i = s'[\ xP + xE_k,\ yP + dy * ( yE_k – i – 1 ) – (yE_k \% 2 )\ ] \quad \text{(G-356)}$$

– The process specified in subclause G.8.7.4.2 is invoked with the sample values $p_i$ and $q_i$ (i = 0..3), chromaEdgeFlag, verticalEdgeFlag, fieldModeFilteringFlag, filterOffsetA, filterOffsetB, $qP_p$, $qP_q$, sliceIdc, fieldMbFlag, mbType, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, sTCoeff, and $rS_L$ as inputs, and the output is assigned to the filtered result sample values $p'_i$ and $q'_i$ with i = 0..2.

– The input sample values $p_i$ and $q_i$ with i = 0..2 are replaced by the corresponding filtered result sample values $p'_i$ and $q'_i$ with i = 0..2 inside the sample array s' as follows.

  – If verticalEdgeFlag is equal to 1,

$$s'[\ xP + xE_k + i,\ yP + dy * yE_k\ ] = q'_i \quad \text{(G-357)}$$
$$s'[\ xP + xE_k – i – 1,\ yP + dy * yE_k\ ] = p'_i \quad \text{(G-358)}$$

  – Otherwise (verticalEdgeFlag is equal to 0),

$$s'[\ xP + xE_k,\ yP + dy * ( yE_k + i ) – ( yE_k \% 2 )\ ] = q'_i \quad \text{(G-359)}$$
$$s'[\ xP + xE_k,\ yP + dy * ( yE_k – i – 1 ) – ( yE_k \% 2 )\ ] = p'_i \quad \text{(G-360)}$$

**G.8.7.4.2 SVC filtering process for a set of samples across a horizontal or vertical block edge**

Inputs to this process are

– the input sample values $p_i$ and $q_i$ with i = 0..3 of a single set of samples across an edge that is to be filtered,

– the variable chromaEdgeFlag,

– the variable verticalEdgeFlag,

– the variable fieldModeFilteringFlag,

– the variables filterOffsetA and filterOffsetB,

– the variables $qP_p$ and $qP_q$,

– the arrays sliceIdc, fieldMbFlag, mbType, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and sTCoeff,

– an array $rS_L$ containing residual sample values.

Outputs of this process are the filtered result sample values $p'_i$ and $q'_i$ with i in the range of 0..2.

The content dependent boundary filtering strength variable bS is derived as follows.

– If chromaEdgeFlag is equal to 0, the SVC derivation process for the luma content dependent boundary filtering strength specified in subclause G.8.7.4.3 is invoked with $p_0$, $q_0$, verticalEdgeFlag, sliceIdc, fieldMbFlag, mbType, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, sTCoeff, and $rS_L$ as inputs, and the output is assigned to bS.

– Otherwise (chromaEdgeFlag is equal to 1), the bS used for filtering a set of samples of a horizontal or vertical chroma edge is set equal to the value of bS for filtering the set of samples of a horizontal or vertical luma edge, respectively, that contains the luma sample at location ( SubWidthC * x, SubHeightC * y ) inside the luma array of the same field, where ( x, y ) is the location of the chroma sample $q_0$ inside the chroma array for that field.

The process specified in subclause 8.7.2.2 is invoked with $p_0$, $q_0$, $p_1$, $q_1$, chromaEdgeFlag, bS, filterOffsetA, filterOffsetB, $qP_p$, and $qP_q$ as inputs, and the output is assigned to filterSamplesFlag, indexA, $\alpha$, and $\beta$.

Depending on the variable filterSamplesFlag, the following applies.

– If filterSamplesFlag is equal to 1, the following applies.

  – If bS is less than 4, the process specified in subclause 8.7.2.3 is invoked with $p_i$ and $q_i$ (i = 0..2), chromaEdgeFlag, bS, $\beta$, and indexA given as input, and the output is assigned to $p'_i$ and $q'_i$ (i = 0..2).

  – Otherwise (bS is equal to 4), the process specified in subclause 8.7.2.4 is invoked with $p_i$ and $q_i$ (i = 0..3), chromaEdgeFlag, $\alpha$, and $\beta$ given as input, and the output is assigned to $p'_i$ and $q'_i$ (i = 0..2).

– Otherwise (filterSamplesFlag is equal to 0), the filtered result samples $p'_i$ and $q'_i$ (i = 0..2) are replaced by the corresponding input samples $p_i$ and $q_i$:

$$\text{for } i = 0..2, \quad p'_i = p_i \tag{G-361}$$
$$\text{for } i = 0..2, \quad q'_i = q_i \tag{G-362}$$

**G.8.7.4.3 SVC derivation process for the luma content dependent boundary filtering strength**

Inputs to this process are

– the input sample values $p_0$ and $q_0$ of a single set of samples across an edge that is to be filtered,

– the variable verticalEdgeFlag,

– the arrays sliceIdc, fieldMbFlag, mbType, predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1, and sTCoeff,

– the array $rS_L$ containing residual sample values.

Output of this process is the variable bS.

The following variables are derived as specified in the following:

– mbAddrP and mbAddrQ specify the macroblocks containing the samples $p_0$ and $q_0$, respectively.

– mbPartIdxP and mbPartIdxQ specify the macroblock partitions containing the samples $p_0$ and $q_0$, respectively.

– subMbPartIdxP and subMbPartIdxQ specify the sub-macroblock partitions containing the samples $p_0$ and $q_0$, respectively.

– pFLXP and pFLXQ with X being replaced by 0 and 1 are equal to predFlagLX[ mbAddrP ][ mbPartIdxP ] and predFlagLX[ mbAddrQ ][ mbPartIdxQ ], respectively.

– refLXP and refFLXQ with X being replaced by 0 and 1 are equal to refIdxLX[ mbAddrP ][ mbPartIdxP ] and refIdxLX[ mbAddrQ ][ mbPartIdxQ ], respectively.

– mvLXP and mvFLXQ with X being replaced by 0 and 1 are equal to mvLX[ mbAddrP ][ mbPartIdxP ][ subMbPartP ] and mvLX[ mbAddrQ ][ mbPartIdxQ ][ subMbPartQ ], respectively.

– numMvP and numMvQ are equal to (pFL0P + pFL1P) and (pFL0Q + pFL1Q), respectively.

– When numMvP and numMvQ are both equal to 1, the variables refX and mvX with X being replaced by P and Q are derived as follows.

  – If pFL0X is equal to 1, refX is set equal to refL0X and mvX is set equal to mvL0X.

– Otherwise (pFL1X is equal to 1), refX is set equal to refL1X and mvX is set equal to mvL1X.

– sliceX with X being replaced by P and Q is the slice with DQId equal to (sliceIdc[ mbAddrP ] & 127) and first_mb_in_slice equal to (sliceIdc[ mbAddrP ] >> 7)

Let the variable mixedModeEdgeFlag be derived as follows.

– If MbaffFrameFlag is equal to 1 and fieldMbFlag[ mbAddrP ] is not equal to fieldMbFlag[ mbAddrQ ], mixedModeEdgeFlag is set equal to 1

– Otherwise, mixedModeEdgeFlag is set equal to 0.

The variable bS is derived as follows.

– If SpatialResolutionChangeFlag is equal to 1 and either or both mbType[ mbAddrP ] or mbType[ mbAddrQ ] is equal to I_BL, the following applies.

  – If either mbType[ mbAddrP ] or mbType[ mbAddrQ ] specifies an Intra macroblock prediction mode other than I_BL, bS is set equal to 4.

  – Otherwise, if mbType[ mbAddrP ] is equal to I_BL and mbType[ mbAddrQ ] is equal to I_BL, the following applies.

    – If the luma transform block of sliceP containing sample $p_0$ or the luma transform block of sliceQ containing sample $q_0$ contains non-zero transform coefficient levels, bS is set equal to 1.

      NOTE 1 – A luma transform block of a layer representation with dependency_id and quality_id is considered to contain non-zero transform coefficient levels, if and only if non-zero transform coefficients are transmitted in the macroblock layer for dependency_id and quality_id for the considered luma transform block. Transform coefficient levels that are transmitted in layers employed by inter-layer prediction are not taken into account.

    – Otherwise, bS is set equal to 0

  – Otherwise (either mbType[ mbAddrP ] or mbType[ mbAddrQ ] specifies an Inter macroblock prediction mode), the following applies.

    – If mbType[ mbAddrP ] specifies an Inter macroblock prediction type and the array $rS_L$ contains any non-zero sample for the luma block containing sample $p_0$, bS is set equal to 2.

    – Otherwise, if mbType[ mbAddrQ ] specifies an Inter macroblock prediction type and the array $rS_L$ contains any non-zero sample for the luma block containing sample $q_0$, bS is set equal to 2.

      NOTE 2 – The array $rS_L$ contains samples for the accumulated residual signal. Transform coefficient values of layer representations that are employed for inter-layer prediction are taken into account.

    – Otherwise, bS is set equal to 1.

– Otherwise, if the block edge is also a macroblock edge and any of the following conditions is true, bS is set equal to 4.

  – fieldMbFlag[ mbAddrP ] is equal to 0 and fieldMbFlag[ mbAddrQ ] is equal to 0 and either or both mbType[ mbAddrP ] or mbType[ mbAddrQ ] specify an Intra macroblock prediction mode.

  – MbaffFrameFlag is equal to 1 or field_pic_flag is equal to 1, and verticalEdgeFlag is equal to 1, and either or both mbType[ mbAddrP ] or mbType[ mbAddrQ ] specify an Intra macroblock prediction mode.

– Otherwise, if any of the following conditions is true, bS is set equal to 3.

  – mixedModeEdgeFlag is equal to 0 and either or both mbType[ mbAddrP ] or mbType[ mbAddrQ ] specify an Intra macroblock prediction mode.

  – mixedModeEdgeFlag is equal to 1, verticalEdgeFlag is equal to 0, and either or both mbType[ mbAddrP ] or mbType[ mbAddrQ ] specify an Intra macroblock prediction mode.

– Otherwise, if any of the following conditions is true, bS is set equal to 2.

  – The array $rS_L$ contains any non-zero sample for the luma block containing sample $p_0$ or the array sTCoeff[ mbAddrP ] contains any non-zero scaled transform coefficient value for the luma block containing sample $p_0$.

  – The array $rS_L$ contains any non-zero sample for the luma block containing sample $q_0$ or the array sTCoeff[ mbAddrQ ] contains any non-zero scaled transform coefficient value for the luma block containing sample $q_0$.

    NOTE 3 – The array $rS_L$ contains samples for the accumulated residual signal. Transform coefficient values of layer representations that are employed for inter-layer prediction are taken into account.

- Otherwise, if any of the following conditions is true, bS is set equal to 1.
  - mixedModeEdgeFlag is equal to 1
  - mixedModeEdgeFlag is equal to 0 and any of the following conditions is true
    - numMvP is not equal to numMvQ
    - numMvP and numMvQ are both equal to 1 and any of the following conditions is true
      - refP and refQ specify different reference pictures
      - the absolute difference between the horizontal or vertical component of the motion vectors mvP and mvQ is greater than or equal to 4 in units of quarter luma frame samples
    - numRefP and numRefQ are both equal to 2 and any of the following conditions is true
      - refL0P and refL1P specify different reference pictures and any of the following conditions is true
        - both of the following conditions are true
          - refL0P and refL0Q specify different reference pictures or refL1P and refL1Q specify different reference pictures
          - refL0P and refL1Q specify different reference pictures or refL1P and refL0Q specify different reference pictures
        - refL0P and refL0Q specifies the same reference picture, refL1P and refL1Q specify the same reference picture, and any of the following conditions is true
          - the absolute difference between the horizontal or vertical component of the motion vectors mvL0P and mvL0Q is greater than or equal to 4 in units of quarter luma frame samples
          - the absolute difference between the horizontal or vertical component of the motion vectors mvL1P and mvL1Q is greater than or equal to 4 in units of quarter luma frame samples
        - refL0P and refL1Q specifies the same reference picture, refL1P and refL0Q specify the same reference picture, and any of the following conditions is true
          - the absolute difference between the horizontal or vertical component of the motion vectors mvL0P and mvL1Q is greater than or equal to 4 in units of quarter luma frame samples
          - the absolute difference between the horizontal or vertical component of the motion vectors mvL1P and mvL0Q is greater than or equal to 4 in units of quarter luma frame samples
      - refL0P and refL1P specify the same reference picture and any of the following conditions is true
        - refL0Q or refL1Q specify a different reference picture than refL0P (or refL1P)
        - refL0Q and refL1Q specify the same reference picture as refL0P (and refL1P) and any of the following conditions is true
          - the absolute difference between the horizontal or vertical component of the motion vectors mvL0P and mvL0Q is greater than or equal to 4 in units of quarter luma frame samples
          - the absolute difference between the horizontal or vertical component of the motion vectors mvL1P and mvL1Q is greater than or equal to 4 in units of quarter luma frame samples
          - the absolute difference between the horizontal or vertical component of the motion vectors mvL0P and mvL1Q is greater than or equal to 4 in units of quarter luma frame samples
          - the absolute difference between the horizontal or vertical component of the motion vectors mvL1P and mvL0Q is greater than or equal to 4 in units of quarter luma frame samples

    NOTE 4 – The determination of whether the reference pictures used for the two macroblock/sub-macroblock partitions are the same or different is based only on which pictures are referenced, without regard to whether a prediction is formed using an index into reference picture list 0 or an index into reference picture list 1, and also without regard to whether the index position within a reference picture list is different.

    NOTE 5 – A vertical difference of 4 in units of quarter luma frame samples is a difference of 2 in units of quarter luma field samples

- Otherwise, bS is set equal to 0.

### G.8.8 Specification of bitstream subsets

Subclause G.8.8.1 specifies the sub-bitstream extraction process.

Subclause G.8.8.2 specifies the base layer bitstream.

#### G.8.8.1 Sub-bitstream extraction process

Inputs to this process are

– a variable pIdTarget (when present),

– a variable tIdTarget (when present),

– a variable dIdTarget (when present),

– a variable qIdTarget (when present).

Output of this process is a sub-bitstream.

When pIdTarget is not present as input to this subclause, pIdTarget is inferred to be equal to 63.

When tIdTarget is not present as input to this subclause, tIdTarget is inferred to be equal to 7.

When dIdTarget is not present as input to this subclause, dIdTarget is inferred to be equal to 7.

When qIdTarget is not present as input to this subclause, qIdTarget is inferred to be equal to 15.

The sub-bitstream is derived by applying the following operations in sequential order.

1. Mark all VCL, filler prefix, and filler data NAL units for which any of the following conditions is true as "to be removed from the bit-stream".
   – priority_id is greater than pIdTarget
   – temporal_id is greater than tIdTarget
   – dependency_id is greater than dIdTarget
   – dependency_id is equal to dIdTarget and quality_id is greater than qIdTarget
2. Remove all access units for which all VCL NAL units are marked as "to be removed from the bit-stream".
3. Remove all VCL, filler prefix, and filler data NAL units that are marked as "to be removed from the bit-stream".
4. When dIdTarget is equal to 0 and qIdTarget is equal to 0, remove the following NAL units.
   – all NAL units with nal_unit_type equal to 14 or 15
   – all NAL units with nal_unit_type equal to 6 in which the first SEI message has payloadType in the range of 24 to 35, inclusive
5. Remove all NAL units with nal_unit_type equal to 6 that only contain SEI messages that are part of a scalable nesting SEI message with any of the following properties
   – sei_temporal_id is greater than tIdTarget
   – the minimum value of ( sei_dependency_id[ i ] << 4 ) + sei_quality_id[ i ] for all i in the range of 0 to num_layer_representations_minus1, inclusive, is greater than ( dIdTarget << 4 ) + qIdTarget

#### G.8.8.2 Specification of the base layer bitstream

Each scalable bitstream that conforms to this specification shall contain a base layer bitstream that conforms to one or more of the profiles specified in Annex A. This base layer bitstream is derived by invoking the sub-bitstream extraction process as specified in subclause G.8.8.1 with dIdTarget being equal to 0 and qIdTarget being equal to 0 and the base layer bitstream being the output.

## G.9 Parsing process

Inputs to this process are bits from the RBSP, a request for a value of a syntax element, and values of prior parsed syntax elements (if applicable).

Output of this process is the value of the syntax element.

This process is invoked for all syntax elements in the syntax tables in subclause G.7.3 with descriptors equal to u(v), ue(v), me(v), se(v), te(v), ce(v), and ae(v).

When parsing coded slice NAL units with nal_unit_type equal to 20, the syntax elements scan_idx_start and scan_idx_end, which are not present in these NAL units, are inferred to be equal to 0 and 15, respectively.

When the parsing process is invoked for the first request for a value of a syntax element in the slice data and entropy_coding_mode_flag is equal to 1, the following ordered steps apply.

1. The initialisation process as specified in subclause 9.3.1 is invoked, where a slice_type equal to EI is interpreted as I, a slice_type equal to EP is interpreted as P, and a slice_type equal to EB is interpreted as B.
2. The initialisation process as specified in subclause G.9.3.1 is invoked.

Depending on entropy_coding_mode_flag and the descriptor, the value of a syntax element is derived as follows.

– If entropy_coding_mode_flag is equal to 0, the following applies.
  – The parsing process for syntax elements coded as coded as ue(v), se(v), or te(v) is specified in subclause 9.1.
  – The parsing process for the syntax element coded_block_pattern is specified in subclause G.9.1.
  – The parsing process for syntax elements of the residual_block_cavlc( ) syntax structure is specified in subclause G.9.2.
– Otherwise (entropy_coding_mode_flag is equal to 1), the value of the syntax element is derived as follows.
  – If the syntax element is equal to base_mode_flag, motion_prediction_flag_l0, motion_prediction_flag_l1, or residual_prediction_flag, the following applies.
    1. The binarization process as specified in subclause G.9.3.2 is invoked.
    2. The decoding process flow as specified in G.9.3.3 is invoked.
  – Otherwise (the syntax element is not equal to base_mode_flag, motion_prediction_flag_l0, motion_prediction_flag_l1, or residual_prediction_flag), the following applies.
    1. The binarization process as specified in subclause 9.3.2 is invoked, where a slice_type equal to EI is interpreted as I, a slice_type equal to EP is interpreted as P, and a slice_type equal to EB is interpreted as B.
    2. The decoding process flow as specified in subclause 9.3.3 is invoked.
    3. When the syntax element is equal to mb_type and the decoded value of mb_type is equal to I_PCM, the arithmetic decoding engine is initialised after decoding of any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data as specified in subclause 9.3.1.2.

### G.9.1 Alternative parsing process for coded block pattern

This process is invoked for the parsing syntax elements with descriptor equal to me(v) when entropy_coding_mode_flag is equal to 0.

Inputs to this process are bits from the RBSP.

Outputs of this process is a value of the syntax element coded_block_pattern.

The parsing process for the syntax elements begins with reading the bits starting at the current location in the bitstream up to and including the first non-zero bit. By counting the number of leading bits that are equal to 0 and assigning this value to the variable leadingZeroBits, the variable codeNum is then derived as follows:

$$codeNum = 2^{leadingZeroBits} - 1 + read\_bits( leadingZeroBits )$$

where the value returned from read_bits( leadingZeroBits ) is interpreted as a binary representation of an unsigned integer with most significant bit written first.

When ref_layer_dq_id is greater than or equal to 0 and ( scan_idx_end – scan_idx_start ) is less than 15, codeNum is set equal to (codeNum – 1).

Depending on codeNum, the following applies.

– If codeNum is equal to –1, the following ordered steps are specified.
  1. The derivation process for neighbouring macroblocks specified in subclause 6.4.10.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

2. When mbAddrN is available, the variable codedBlockPatternN is set equal to (16 * cbpChromaN + cbpLumaN) with cbpChromaN and cbpLumaN representing the values of CodedBlockPatternLuma and CodedBlockPatternChroma for the macroblock mbAddrN (with N being either A or B).

3. Depending on mbAddrA and mbAddrB, the following applies.

   – If mbAddrA is available, coded_block_pattern is set equal to codedBlockPatternA.

   – Otherwise, if mbAddrB is available, coded_block_pattern is set equal to codedBlockPatternB.

   – Otherwise (mbAddrA and mbAddrB are not available), coded_block_pattern is set equal to 0.

– Otherwise (codeNum is greater than or equal to 0), the mapping process for coded block pattern as specified in subclause 9.1.2 is invoked with codeNum as input and the output is assigned to the syntax element coded_block_pattern.

### G.9.2 Alternative CAVLC parsing process for transform coefficient levels

This process is invoked for the parsing syntax elements with descriptor equal to ce(v) when entropy_coding_mode_flag is equal to 0.

Inputs to this process are a request for a value of a syntax element, bits from slice data, a maximum number of non-zero transform coefficient levels maxNumCoeff, the luma block index luma4x4BlkIdx or the chroma block index chroma4x4BlkIdx, cb4x4BlkIdx or cr4x4BlkIdx of the current block of transform coefficient levels.

Output of this process is the list coeffLevel containing transform coefficient levels of the luma block with block index luma4x4BlkIdx or the chroma block with block index chroma4x4BlkIdx, cb4x4BlkIdx or cr4x4BlkIdx.

The process is specified in the following ordered steps:

1. All transform coefficient levels, with indices from 0 to maxNumCoeff – 1, in the list coeffLevel are set equal to 0.

2. The total number of non-zero transform coefficient levels TotalCoeff( coeff_token ) and the number of trailing one transform coefficient levels TrailingOnes( coeff_token ) are derived by parsing coeff_token as follows.

   a. The parsing process of coeff_token as specified in subclause 9.2.1 is invoked and the outputs are TotalCoeff( coeff_token ), TrailingOnes( coeff_token ), and nC.

   b. When the CAVLC parsing process not invoked for ChromaDCLevel and ( scan_idx_end - scan_idx_start ) is less than 15, nC modified by setting it equal to Min( 7, nC ), and the additional parsing process for total number of transform coefficient levels and trailing ones as specified in subclause G.9.2.1 is invoked with nC, totalCoeffStart set equal to TotalCoeff( coeff_token ), and trailingOnesStart set equal to TrailingOnes( coeff_token ) as the inputs and the outputs are assigned to TotalCoeff( coeff_token ) and TrailingOnes( coeff_token ).

3. When TotalCoeff( coeff_token ) is greater than 0, the following applies.

   a. The non-zero transform coefficient levels are derived by parsing trailing_ones_sign_flag, level_prefix, and level_suffix as specified in subclause 9.2.2.

   b. The runs of zero transform coefficient levels before each non-zero transform coefficient level are derived by parsing total_zeros and run_before as specified in subclause G.9.2.2.

   c. The level and run information are combined into the list coeffLevel as specified in subclause 9.2.4.

#### G.9.2.1 Additional parsing process for total number of transform coefficient levels and trailing ones

Inputs to this process are variables nC, totalCoeffStart, and trailingOnesStart.

Outputs of this process are variables totalCoeff and trailingOnes.

The value of totalCoeffStart resulting from decoding coeff_token as specified in subclause 9.2.1 shall be in the range of 0 to ( scan_idx_end – scan_idx_start + 1 ), inclusive.

Let invTotalCoeff( coeffTokenIdx ) and invTrailingOnes( coeffTokenIdx) be functions that map the variable coeffTokenIdx to the variables nX and nY, respectively, as specified by Table G-9 given the variable nC.

A variable targetCoeffTokenIdx is derived as specified by Table G-9 given the variables nC, nX = totalCoeffStart, and nY = trailingOnesStart.

A variable coeffTokenIdx is derived as specified by the following pseudo code.

```
for( coeffTokenIdx = 0, i = 0; i <= targetCoeffTokenIdx; coeffTokenIdx++ )
{
  if( invTotalCoeff  ( coeffTokenIdx ) < min(  4, scan_idx_end – scan_idx_start + 2 ) &&
      invTrailingOnes( coeffTokenIdx ) < min( 17, scan_idx_end – scan_idx_start + 2 ) )
    i++;
}
```

The variable totalCoeff is set equal to invTotalCoeff( coeffTokenIdx – 1 ) and the variable trailingOnes is set equal to invTrailingOnes( coeffTokenIdx – 1 ).

**Table G-9 – Mapping of ( nX, nY ) to coeffTokenIdx and vice versa**

| nY | nX | 0 <= nC < 2 | 2 <= nC < 4 | 4 <= nC < 8 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 4 | 7 | 16 |
| 1 | 1 | 1 | 1 | 1 |
| 0 | 2 | 9 | 11 | 20 |
| 1 | 2 | 5 | 5 | 8 |
| 2 | 2 | 2 | 2 | 2 |
| 0 | 3 | 13 | 15 | 23 |
| 1 | 3 | 10 | 8 | 11 |
| 2 | 3 | 7 | 9 | 9 |
| 3 | 3 | 3 | 3 | 3 |
| 0 | 4 | 17 | 19 | 24 |
| 1 | 4 | 14 | 12 | 13 |
| 2 | 4 | 11 | 13 | 12 |
| 3 | 4 | 6 | 4 | 4 |
| 0 | 5 | 21 | 22 | 28 |
| 1 | 5 | 18 | 16 | 15 |
| 2 | 5 | 15 | 17 | 14 |
| 3 | 5 | 8 | 6 | 5 |
| 0 | 6 | 25 | 23 | 30 |
| 1 | 6 | 22 | 20 | 17 |
| 2 | 6 | 19 | 21 | 18 |
| 3 | 6 | 12 | 10 | 6 |
| 0 | 7 | 29 | 27 | 31 |
| 1 | 7 | 26 | 24 | 21 |
| 2 | 7 | 23 | 25 | 22 |
| 3 | 7 | 16 | 14 | 7 |
| 0 | 8 | 32 | 31 | 32 |
| 1 | 8 | 30 | 28 | 25 |
| 2 | 8 | 27 | 29 | 26 |
| 3 | 8 | 20 | 18 | 10 |
| 0 | 9 | 33 | 35 | 36 |