

VIDEO PROCESSING USING SYNTAX ELEMENTS

TECHNICAL FIELD

[001] Disclosed are embodiments related to the processing (e.g., decoding) of video data.

BACKGROUND

[002] 1. HEVC and VVC

[003] High Efficiency Video Coding (HEVC) is a block-based video codec standardized by ITU-T and MPEG that utilizes both temporal and spatial prediction. Spatial prediction is achieved using intra (I) prediction from within the current picture. Temporal prediction is achieved using uni-directional (P) or bi-directional inter (B) prediction on block level from previously decoded reference pictures. In the encoder, the difference between the original pixel data and the predicted pixel data, referred to as the residual, is transformed into the frequency domain, quantized and then entropy coded before transmitted together with necessary prediction parameters such as prediction mode and motion vectors, also entropy coded. The decoder performs entropy decoding, inverse quantization and inverse transformation to obtain the residual, and then adds the residual to an intra or inter prediction to reconstruct a picture.

[004] MPEG and ITU-T is working on the successor to HEVC within the Joint Video Exploratory Team (JVET). The name of this video codec under development is Versatile Video Coding (VVC). The current version of the VVC draft specification at the time of writing this text is JVET-Q2001-vD.

[005] 2. Components

[006] A video (a.k.a., video sequence) consists of a series of pictures (a.k.a., images) where each picture consists of one or more components. Each component can be described as a two-dimensional rectangular array of sample values. It is common that a picture in a video sequence consists of three components; one luma component Y where the sample values are luma values and two chroma components Cb and Cr, where the sample values are chroma values. It is also common that the dimensions of the chroma components are smaller than the luma components

by a factor of two in each dimension. For example, the size of the luma component of an HD picture would be 1920x1080 and the chroma components would each have the dimension of 960x540. Components are sometimes referred to as color components.

[007] 3. Blocks and units

[008] A block is one two-dimensional array of samples. In video coding, each component is split into blocks and the coded video bitstream consists of a series of coded blocks. It is common in video coding that the picture is split into units that cover a specific area of the picture. Each unit consists of all blocks from all components that make up that specific area and each block belongs fully to one unit. The macroblock in H.264 and the Coding unit (CU) in HEVC are examples of units.

[009] A block can alternatively be defined as a two-dimensional array that a transform used in coding is applied to. These blocks are known under the name “transform blocks”. Alternatively, a block can be defined as a two-dimensional array that a single prediction mode is applied to. These blocks can be called “prediction blocks”. In this application, the word block is not tied to one of these definitions but that the descriptions herein can apply to either definition.

[0010] 4. Residuals, transforms and quantization

[0011] A residual block consists of samples that represents sample value differences between sample values of the original source blocks and the prediction blocks. The residual block is processed using a spatial transform. In the encoder, the transform coefficients are quantized according to a quantization parameter (QP) which controls the precision of the quantized coefficients. The quantized coefficients can be referred to as residual coefficients. A high QP value would result in low precision of the coefficients and thus low fidelity of the residual block. A decoder receives the residual coefficients, applies inverse quantization and inverse transform to derive the residual block.

[0012] 5. NAL units

[0013] Both HEVC and VVC define a Network Abstraction Layer (NAL). All the data (i.e. both Video Coding Layer (VCL) or non-VCL data) in HEVC and VVC is encapsulated in NAL units. A VCL NAL unit contains data that represents picture sample values. A non-VCL NAL

unit contains additional associated data such as parameter sets and supplemental enhancement information (SEI) messages. The NAL unit in HEVC begins with a header which specifies the NAL unit type of the NAL unit that identifies what type of data is carried in the NAL unit, the layer ID and the temporal ID for which the NAL unit belongs to. The NAL unit type is transmitted in the `nal_unit_type` codeword in the NAL unit header and the type indicates and defines how the NAL unit should be parsed and decoded. The rest of the bytes of the NAL unit is payload of the type indicated by the NAL unit type. A bitstream consists of a series of concatenated NAL units.

[0014] The syntax for the NAL unit header for HEVC is shown in Table 1.

TABLE 1 – HEVC NAL unit header syntax

<code>nal_unit_header() {</code>	Descriptor
<code>forbidden_zero_bit</code>	<code>f(1)</code>
<code>nal_unit_type</code>	<code>u(6)</code>
<code>nuh_layer_id</code>	<code>u(6)</code>
<code>nuh_temporal_id_plus1</code>	<code>u(3)</code>
<code>}</code>	

[0015] The syntax for the NAL unit header in the current version of the VVC draft is shown in Table 2.

TABLE 2 – VVC NAL unit header syntax

<code>nal_unit_header() {</code>	Descriptor
<code>forbidden_zero_bit</code>	<code>f(1)</code>
<code>nuh_reserved_zero_bit</code>	<code>u(1)</code>
<code>nuh_layer_id</code>	<code>u(6)</code>
<code>nal_unit_type</code>	<code>u(5)</code>
<code>nuh_temporal_id_plus1</code>	<code>u(3)</code>
<code>}</code>	

[0016] The NAL unit types of the current VVC draft are shown in Table 3. The decoding order is the order in which NAL units shall be decoded, which is the same as the order of the NAL units within the bitstream. The decoding order may be different from the output order, which is the order in which decoded pictures are to be output, such as for display, by the decoder.

TABLE 3 – NAL unit types in VVC

nal_unit_type	Name of nal_unit_type	Content of NAL unit and RBSP syntax structure	NAL unit type class
0	TRAIL_NUT	Coded slice of a trailing picture slice_layer_rbsp()	VCL
1	STSA_NUT	Coded slice of an STSA picture slice_layer_rbsp()	VCL
2	RADL_NUT	Coded slice of a RADL picture slice_layer_rbsp()	VCL
3	RASL_NUT	Coded slice of a RASL picture slice_layer_rbsp()	VCL
4..6	RSV_VCL_4.. RSV_VCL_6	Reserved non-IRAP VCL NAL unit types	VCL
7 8	IDR_W_RADL IDR_N_LP	Coded slice of an IDR picture slice_layer_rbsp()	VCL
9	CRA_NUT	Coded slice of a CRA picture silce_layer_rbsp()	VCL
10	GDR_NUT	Coded slice of a GDR picture slice_layer_rbsp()	VCL
11 12	RSV_IRAP_11 RSV_IRAP_12	Reserved IRAP VCL NAL unit types	VCL
13	DCI_NUT	Decoding capability information decoding_capability_information_rbsp()	non-VCL
14	VPS_NUT	Video parameter set video_parameter_set_rbsp()	non-VCL
15	SPS_NUT	Sequence parameter set seq_parameter_set_rbsp()	non-VCL
16	PPS_NUT	Picture parameter set pic_parameter_set_rbsp()	non-VCL
17 18	PREFIX_APS_NUT SUFFIX_APS_NUT	Adaptation parameter set adaptation_parameter_set_rbsp()	non-VCL
19	PH_NUT	Picture header picture_header_rbsp()	non-VCL
20	AUD_NUT	AU delimiter access_unit_delimiter_rbsp()	non-VCL

21	EOS_NUT	End of sequence end_of_seq_rbsp()	non-VCL
22	EOB_NUT	End of bitstream end_of_bitstream_rbsp()	non-VCL
23 24	PREFIX_SEI_NUT SUFFIX_SEI_NUT	Supplemental enhancement information sei_rbsp()	non-VCL
25	FD_NUT	Filler data filler_data_rbsp()	non-VCL
26 27	RSV_NVCL_26 RSV_NVCL_27	Reserved non-VCL NAL unit types	non-VCL
28..31	UNSPEC_28.. UNSPEC_31	Unspecified non-VCL NAL unit types	non-VCL

[0017] 6. Slices

[0018] The concept of slices in HEVC divides the picture into independently coded slices, where decoding of one slice in a picture is independent of other slices of the same picture. Different coding types could be used for slices of the same picture, i.e. a slice could either be an I-slice, P-slice or B-slice. One purpose of slices is to enable resynchronization in case of data loss. In HEVC, a slice is a set of CTUs.

[0019] In the current version of VVC, a picture may be partitioned into either raster scan slices or rectangular slices. A raster scan slice consists of a number of complete tiles in raster scan order. A rectangular slice consists of a group of tiles that together occupy a rectangular region in the picture or a consecutive number of CTU rows inside one tile. Each slice has a slice header comprising syntax elements. Decoded slice header values from these syntax elements are used when decoding the slice. Each slice is carried in one VCL NAL unit. In a previous version of the VVC draft specification, slices were referred to as tile groups.

[0020] 7. Picture header

[0021] The current VVC draft includes a picture header (PH), which is a non-VCL NAL unit having nal_unit_type equal to PH_NUT. The picture header is similar to the slice header, but the values of the syntax elements in the picture header are used to decode all slices of one picture.

[0022] In the current version of VVC, the VCL NAL units of a coded picture may be preceded by a picture header. The picture header contains parameters that are common for all slices of the associated picture. Some of the parameters may be signaled in either the picture header or in the slice header of each slice in the picture conditioned on a presence flag signaled in the PPS. HEVC does not have picture headers. In the current version of VVC, either there is a picture header NAL unit for each picture, or the picture header syntax is present in the slice header. In the latter case, only one slice per picture is allowed. The picture header and picture header structure syntax in the current version of VVC is shown below in tables 4 and 5.

TABLE 4 – picture header syntax in VVC

picture_header_rbsp() {	Descriptor
picture_header_structure()	
rbsp_trailing_bits()	
}	

TABLE 5 – picture header structure syntax in VVC

picture_header_structure() {	Descriptor
gdr_or_irap_pic_flag	u(1)
if(gdr_or_irap_pic_flag)	
gdr_pic_flag	u(1)
ph_inter_slice_allowed_flag	u(1)
if(ph_inter_slice_allowed_flag)	
ph_intra_slice_allowed_flag	u(1)
non_reference_picture_flag	u(1)
ph_pic_parameter_set_id	ue(v)
ph_pic_order_cnt_lsb	u(v)
if(gdr_or_irap_pic_flag)	
no_output_of_prior_pics_flag	u(1)
if(gdr_pic_flag)	
recovery_poc_cnt	ue(v)
for(i = 0; i < NumExtraPhBits; i++)	
ph_extra_bit[i]	u(1)
if(sps_poc_msb_flag) {	
ph_poc_msb_present_flag	u(1)
if(ph_poc_msb_present_flag)	
poc_msb_val	u(v)
}	
}	

if(sps_alf_enabled_flag && alf_info_in_ph_flag) {	
ph_alf_enabled_flag	u(1)
if(ph_alf_enabled_flag) {	
ph_num_alf_aps_ids_luma	u(3)
for(i = 0; i < ph_num_alf_aps_ids_luma; i++)	
ph_alf_aps_id_luma[i]	u(3)
if(ChromaArrayType != 0)	
ph_alf_chroma_idc	u(2)
if(ph_alf_chroma_idc > 0)	
ph_alf_aps_id_chroma	u(3)
if(sps_ccalf_enabled_flag) {	
ph_cc_alf_cb_enabled_flag	u(1)
if(ph_cc_alf_cb_enabled_flag)	
ph_cc_alf_cb_aps_id	u(3)
ph_cc_alf_cr_enabled_flag	u(1)
if(ph_cc_alf_cr_enabled_flag)	
ph_cc_alf_cr_aps_id	u(3)
}	
}	
}	
if(sps_lmcs_enabled_flag) {	
ph_lmcs_enabled_flag	u(1)
if(ph_lmcs_enabled_flag) {	
ph_lmcs_aps_id	u(2)
if(ChromaArrayType != 0)	
ph_chroma_residual_scale_flag	u(1)
}	
}	
if(sps_scaling_list_enabled_flag) {	
ph_scaling_list_present_flag	u(1)
if(ph_scaling_list_present_flag)	
ph_scaling_list_aps_id	u(3)
}	
if(sps_virtual_boundaries_enabled_flag && !sps_virtual_boundaries_present_flag) {	
ph_virtual_boundaries_present_flag	u(1)
if(ph_virtual_boundaries_present_flag) {	
ph_num_ver_virtual_boundaries	u(2)
for(i = 0; i < ph_num_ver_virtual_boundaries; i++)	
ph_virtual_boundaries_pos_x[i]	u(13)
ph_num_hor_virtual_boundaries	u(2)
for(i = 0; i < ph_num_hor_virtual_boundaries; i++)	

ph_virtual_boundaries_pos_y[i]	u(13)
}	
}	
if(output_flag_present_flag)	
pic_output_flag	u(1)
if(rpl_info_in_ph_flag)	
ref_pic_lists()	
if(partition_constraints_override_enabled_flag)	
partition_constraints_override_flag	u(1)
if(ph_intra_slice_allowed_flag) {	
if(partition_constraints_override_flag) {	
ph_log2_diff_min_qt_min_cb_intra_slice_luma	ue(v)
ph_max_mtt_hierarchy_depth_intra_slice_luma	ue(v)
if(ph_max_mtt_hierarchy_depth_intra_slice_luma != 0) {	
ph_log2_diff_max_bt_min_qt_intra_slice_luma	ue(v)
ph_log2_diff_max_tt_min_qt_intra_slice_luma	ue(v)
}	
if(qtbtt_dual_tree_intra_flag) {	
ph_log2_diff_min_qt_min_cb_intra_slice_chroma	ue(v)
ph_max_mtt_hierarchy_depth_intra_slice_chroma	ue(v)
if(ph_max_mtt_hierarchy_depth_intra_slice_chroma != 0) {	
ph_log2_diff_max_bt_min_qt_intra_slice_chroma	ue(v)
ph_log2_diff_max_tt_min_qt_intra_slice_chroma	ue(v)
}	
}	
}	
if(cu_qp_delta_enabled_flag)	
ph_cu_qp_delta_subdiv_intra_slice	ue(v)
if(pps_cu_chroma_qp_offset_list_enabled_flag)	
ph_cu_chroma_qp_offset_subdiv_intra_slice	ue(v)
}	
if(ph_inter_slice_allowed_flag) {	
if(partition_constraints_override_flag) {	
ph_log2_diff_min_qt_min_cb_inter_slice	ue(v)
ph_max_mtt_hierarchy_depth_inter_slice	ue(v)
if(ph_max_mtt_hierarchy_depth_inter_slice != 0) {	
ph_log2_diff_max_bt_min_qt_inter_slice	ue(v)
ph_log2_diff_max_tt_min_qt_inter_slice	ue(v)
}	
}	
if(cu_qp_delta_enabled_flag)	
ph_cu_qp_delta_subdiv_inter_slice	ue(v)

if(pps_cu_chroma_qp_offset_list_enabled_flag)	
ph_cu_chroma_qp_offset_subdiv_inter_slice	ue(v)
if(sps_temporal_mvp_enabled_flag) {	
ph_temporal_mvp_enabled_flag	u(1)
if(ph_temporal_mvp_enabled_flag && rpl_info_in_ph_flag) {	
ph_collocated_from_l0_flag	u(1)
if((ph_collocated_from_l0_flag && num_ref_entries[0][RplIdx[0]] > 1) (!ph_collocated_from_l0_flag && num_ref_entries[1][RplIdx[1]] > 1))	
ph_collocated_ref_idx	ue(v)
}	
}	
mvd_l1_zero_flag	u(1)
if(sps_fpel_mmvd_enabled_flag)	
ph_fpel_mmvd_enabled_flag	u(1)
if(sps_bdof_pic_present_flag)	
ph_disable_bdof_flag	u(1)
if(sps_dmvr_pic_present_flag)	
ph_disable_dmvr_flag	u(1)
if(sps_prof_pic_present_flag)	
ph_disable_prof_flag	u(1)
if((pps_weighted_pred_flag pps_weighted_bipred_flag) && wp_info_in_ph_flag)	
pred_weight_table()	
}	
if(qp_delta_info_in_ph_flag)	
ph_qp_delta	se(v)
if(sps_joint_cbr_enabled_flag)	
ph_joint_cbr_sign_flag	u(1)
if(sps_sao_enabled_flag && sao_info_in_ph_flag) {	
ph_sao_luma_enabled_flag	u(1)
if(ChromaArrayType != 0)	
ph_sao_chroma_enabled_flag	u(1)
}	
if(sps_dep_quant_enabled_flag)	
ph_dep_quant_enabled_flag	u(1)
if(sps_sign_data_hiding_enabled_flag && !ph_dep_quant_enabled_flag)	
pic_sign_data_hiding_enabled_flag	u(1)
if(deblocking_filter_override_enabled_flag && dbf_info_in_ph_flag) {	
ph_deblocking_filter_override_flag	u(1)
if(ph_deblocking_filter_override_flag) {	

ph_deblocking_filter_disabled_flag	u(1)
if(!ph_deblocking_filter_disabled_flag) {	
ph_beta_offset_div2	se(v)
ph_tc_offset_div2	se(v)
ph_cb_beta_offset_div2	se(v)
ph_cb_tc_offset_div2	se(v)
ph_cr_beta_offset_div2	se(v)
ph_cr_tc_offset_div2	se(v)
}	
}	
}	
if(picture_header_extension_present_flag) {	
ph_extension_length	ue(v)
for(i = 0; i < ph_extension_length; i++)	
ph_extension_data_byte[i]	u(8)
}	
}	

[0023] 8. Slice header

[0024] The current VVC draft includes, just like HEVC, a slice header (SH) for each slice. The slice header comprises parameters that are common for all tiles or CTU rows within a tile represented in the slice. In the current version of VVC, some header parameters may be either signaled in the picture header or in the slice header, conditioned on a presence flag signaled in the picture parameter set. In the current version of VVC, the picture header parameters may be signaled in the slice header conditioned on a flag `picture_header_in_slice_header_flag` which is signaled in the slice header. The slice header syntax in the current version of VVC is shown below in table 6.

TABLE 6 – slice header syntax in VVC

<code>slice_header() {</code>	Descriptor
picture_header_in_slice_header_flag	u(1)
if(picture_header_in_slice_header_flag)	
picture_header_structure()	
if(subpic_info_present_flag)	
slice_subpic_id	u(v)
if((rect_slice_flag && NumSlicesInSubpic[CurrSubpicIdx] > 1) (!rect_slice_flag && NumTilesInPic > 1))	
slice_address	u(v)

for(i = 0; i < NumExtraPhBits; i++)	
sh_extra_bit[i]	u(1)
if(!rect_slice_flag && NumTilesInPic > 1)	
num_tiles_in_slice_minus1	ue(v)
if(ph_inter_slice_allowed_flag)	
slice_type	ue(v)
if(sps_alf_enabled_flag && !alf_info_in_ph_flag) {	
slice_alf_enabled_flag	u(1)
if(slice_alf_enabled_flag) {	
slice_num_alf_aps_ids_luma	u(3)
for(i = 0; i < slice_num_alf_aps_ids_luma; i++)	
slice_alf_aps_id_luma[i]	u(3)
if(ChromaArrayType != 0)	
slice_alf_chroma_idc	u(2)
if(slice_alf_chroma_idc)	
slice_alf_aps_id_chroma	u(3)
if(sps_ccalf_enabled_flag) {	
slice_cc_alf_cb_enabled_flag	u(1)
if(slice_cc_alf_cb_enabled_flag)	
slice_cc_alf_cb_aps_id	u(3)
slice_cc_alf_cr_enabled_flag	u(1)
if(slice_cc_alf_cr_enabled_flag)	
slice_cc_alf_cr_aps_id	u(3)
}	
}	
if(separate_colour_plane_flag == 1)	
colour_plane_id	u(2)
if(!rpl_info_in_ph_flag && ((nal_unit_type != IDR_W_RADL && nal_unit_type != IDR_N_LP) sps_idr_rpl_present_flag))	
ref_pic_lists()	
if((rpl_info_in_ph_flag ((nal_unit_type != IDR_W_RADL && nal_unit_type != IDR_N_LP) sps_idr_rpl_present_flag)) && (slice_type != 1 && num_ref_entries[0][RplIdx[0]] > 1) (slice_type == B && num_ref_entries[1][RplIdx[1]] > 1)) {	
num_ref_idx_active_override_flag	u(1)
if(num_ref_idx_active_override_flag)	
for(i = 0; i < (slice_type == B ? 2 : 1); i++)	
if(num_ref_entries[i][RplIdx[i]] > 1)	
num_ref_idx_active_minus1[i]	ue(v)
}	

if(slice_type != I) {	
if(cabac_init_present_flag)	
cabac_init_flag	u(1)
if(ph_temporal_mvp_enabled_flag && !rpl_info_in_ph_flag) {	
if(slice_type == B)	
slice_collocated_from_I0_flag	u(1)
if((slice_collocated_from_I0_flag && NumRefIdxActive[0] > 1) (! slice_collocated_from_I0_flag && NumRefIdxActive[1] > 1))	
slice_collocated_ref_idx	ue(v)
}	
if(!wp_info_in_ph_flag && ((pps_weighted_pred_flag && slice_type == P) (pps_weighted_bipred_flag && slice_type == B)))	
pred_weight_table()	
}	
if(!qp_delta_info_in_ph_flag)	
slice_qp_delta	se(v)
if(pps_slice_chroma_qp_offsets_present_flag) {	
slice_cb_qp_offset	se(v)
slice_cr_qp_offset	se(v)
if(sps_joint_cbr_enabled_flag)	
slice_joint_cbr_qp_offset	se(v)
}	
if(pps_cu_chroma_qp_offset_list_enabled_flag)	
cu_chroma_qp_offset_enabled_flag	u(1)
if(sps_sao_enabled_flag && !sao_info_in_ph_flag) {	
slice_sao_luma_flag	u(1)
if(ChromaArrayType != 0)	
slice_sao_chroma_flag	u(1)
}	
if(deblocking_filter_override_enabled_flag && !dbf_info_in_ph_flag)	
slice_deblocking_filter_override_flag	u(1)
if(slice_deblocking_filter_override_flag) {	
slice_deblocking_filter_disabled_flag	u(1)
if(!slice_deblocking_filter_disabled_flag) {	
slice_beta_offset_div2	se(v)
slice_tc_offset_div2	se(v)
slice_cb_beta_offset_div2	se(v)
slice_cb_tc_offset_div2	se(v)
slice_cr_beta_offset_div2	se(v)
slice_cr_tc_offset_div2	se(v)
}	

}	
slice_ts_residual_coding_disabled_flag	u(1)
if(ph_lmcs_enabled_flag)	
slice_lmcs_enabled_flag	u(1)
if(pic_scaling_list_enabled_flag)	
slice_scaling_list_present_flag	u(1)
if(NumEntryPoints > 0) {	
offset_len_minus1	ue(v)
for(i = 0; i < NumEntryPoints; i++)	
entry_point_offset_minus1[i]	u(v)
}	
if(slice_header_extension_present_flag) {	
slice_header_extension_length	ue(v)
for(i = 0; i < slice_header_extension_length; i++)	
slice_header_extension_data_byte[i]	u(8)
}	
byte_alignment()	
}	

[0025] 9. Parameter Sets

[0026] HEVC specifies three types of parameter sets, the picture parameter set (PPS), the sequence parameter set (SPS) and the video parameter set (VPS). The PPS contains data that is common for a whole picture, the SPS contains data that is common for a coded video sequence (CVS) and the VPS contains data that is common for multiple CVSs.

[0027] VVC also uses these parameter set types. In VVC, there is also the adaptation parameter set (APS) and the decoding capability information (DCI) NAL unit. The APS may contain information that can be used for multiple slices and two slices of the same picture can use different APSes. The DCI consists of information specifying the “worst case” in terms of profile and level that the decoder will encounter in the entire bitstream. The picture parameter set syntax in the current version of VVC is shown below.

TABLE 7 – PPS syntax

pic_parameter_set_rbsp() {	Descriptor
pps_pic_parameter_set_id	ue(v)
pps_seq_parameter_set_id	u(4)

mixed_nalu_types_in_pic_flag	u(1)
pic_width_in_luma_samples	ue(v)
pic_height_in_luma_samples	ue(v)
pps_conformance_window_flag	u(1)
if(pps_conformance_window_flag) {	
pps_conf_win_left_offset	ue(v)
pps_conf_win_right_offset	ue(v)
pps_conf_win_top_offset	ue(v)
pps_conf_win_bottom_offset	ue(v)
}	
scaling_window_explicit_signalling_flag	u(1)
if(scaling_window_explicit_signalling_flag) {	
scaling_win_left_offset	ue(v)
scaling_win_right_offset	ue(v)
scaling_win_top_offset	ue(v)
scaling_win_bottom_offset	ue(v)
}	
output_flag_present_flag	u(1)
subpic_id_mapping_in_pps_flag	u(1)
if(subpic_id_mapping_in_pps_flag) {	
pps_num_subpics_minus1	ue(v)
pps_subpic_id_len_minus1	ue(v)
for(i = 0; i <= pps_num_subpic_minus1; i++)	
pps_subpic_id[i]	u(v)
}	
no_pic_partition_flag	u(1)
if(!no_pic_partition_flag) {	
pps_log2_ctu_size_minus5	u(2)
num_exp_tile_columns_minus1	ue(v)
num_exp_tile_rows_minus1	ue(v)
for(i = 0; i <= num_exp_tile_columns_minus1; i++)	
tile_column_width_minus1[i]	ue(v)
for(i = 0; i <= num_exp_tile_rows_minus1; i++)	
tile_row_height_minus1[i]	ue(v)
if(NumTilesInPic > 1)	
rect_slice_flag	u(1)
if(rect_slice_flag)	
single_slice_per_subpic_flag	u(1)
if(rect_slice_flag && !single_slice_per_subpic_flag) {	
num_slices_in_pic_minus1	ue(v)
if(num_slices_in_pic_minus1 > 0)	
tile_idx_delta_present_flag	u(1)

for(i = 0; i < num_slices_in_pic_minus1; i++) {	
if(NumTileColumns > 1)	
slice_width_in_tiles_minus1[i]	ue(v)
if(NumTileRows > 1 && (tile_idx_delta_present_flag tileidx % NumTileColumns == 0))	
slice_height_in_tiles_minus1[i]	ue(v)
if(slice_width_in_tiles_minus1[i] == 0 && slice_height_in_tiles_minus1[i] == 0 && RowHeight[SliceTopLeftTileIdx[i] / NumTileColumns] > 1) {	
num_exp_slices_in_tile[i]	ue(v)
numExpSlicesInTile = num_exp_slices_in_tile[i]	
for(j = 0; j < numExpSlicesInTile; j++)	
exp_slice_height_in_ctus_minus1[j]	ue(v)
i += NumSlicesInTile[i]	
}	
if(tile_idx_delta_present_flag && i < num_slices_in_pic_minus1)	
tile_idx_delta[i]	se(v)
}	
}	
loop_filter_across_tiles_enabled_flag	u(1)
loop_filter_across_slices_enabled_flag	u(1)
}	
cabac_init_present_flag	u(1)
for(i = 0; i < 2; i++)	
num_ref_idx_default_active_minus1[i]	ue(v)
rpl1_idx_present_flag	u(1)
init_qp_minus26	se(v)
cu_qp_delta_enabled_flag	u(1)
pps_chroma_tool_offsets_present_flag	u(1)
if(pps_chroma_tool_offsets_present_flag) {	
pps_cb_qp_offset	se(v)
pps_cr_qp_offset	se(v)
pps_joint_cbcr_qp_offset_present_flag	u(1)
if(pps_joint_cbcr_qp_offset_present_flag)	
pps_joint_cbcr_qp_offset_value	se(v)
pps_slice_chroma_qp_offsets_present_flag	u(1)
pps_cu_chroma_qp_offset_list_enabled_flag	u(1)
}	
if(pps_cu_chroma_qp_offset_list_enabled_flag) {	
chroma_qp_offset_list_len_minus1	ue(v)
for(i = 0; i <= chroma_qp_offset_list_len_minus1; i++) {	
cb_qp_offset_list[i]	se(v)

cr_qp_offset_list[i]	se(v)
if(pps_joint_cbr_qp_offset_present_flag)	
joint_cbr_qp_offset_list[i]	se(v)
}	
}	
pps_weighted_pred_flag	u(1)
pps_weighted_bipred_flag	u(1)
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
pps_deblocking_filter_disabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
rpl_info_in_ph_flag	u(1)
if(deblocking_filter_override_enabled_flag)	
dbf_info_in_ph_flag	u(1)
sao_info_in_ph_flag	u(1)
alf_info_in_ph_flag	u(1)
if((pps_weighted_pred_flag pps_weighted_bipred_flag) && rpl_present_in_ph_flag)	
wp_info_in_ph_flag	u(1)
qp_delta_info_in_ph_flag	u(1)
pps_ref_wraparound_enabled_flag	u(1)
if(pps_ref_wraparound_enabled_flag)	
pps_ref_wraparound_offset	ue(v)
picture_header_extension_present_flag	u(1)
slice_header_extension_present_flag	u(1)
pps_extension_flag	u(1)
if(pps_extension_flag)	
while(more_rbsp_data())	
pps_extension_data_flag	u(1)
rbsp_trailing_bits()	
}	

[0028] 2.1.10 VVC block structure

[0029] The draft VVC video coding standard uses a block structure referred to as quadtree plus binary tree plus ternary tree block structure (QTBT+TT) where each picture is first partitioned into square blocks called coding tree units (CTU). The size of all CTUs are identical. Each CTU is further partitioned into coding units (CU) that can have either square or rectangular shapes. The CTU is first partitioned by a quad tree structure, then it may be further partitioned with equally sized partitions either vertically or horizontally in a binary structure to form coding units (CUs). A block could thus have either a square or rectangular shape. The depth of the quad tree and binary tree can be set by the encoder in the bitstream. An example of dividing a CTU using QTBT is illustrated in FIG. 1. That is, FIG. 1 shows partitioning a CTU into CUs using QTBT. The ternary tree (TT) part adds the possibility to divide a CU into three partitions instead of two equally sized partitions; this increases the possibilities to use a block structure that better fits the content structure in a picture.

[0030] 11. Deblocking filter

[0031] After blocks have been reconstructed, deblocking is applied to reduce boundary artifacts between coded blocks.

[0032] Deblocking is first applied on vertical boundaries and then on horizontal boundaries. The boundaries are either transform block boundaries or prediction block boundaries. A deblocking filter boundary strength parameter (bS) is set for each boundary. If the value of bS is larger than 0, then deblocking may be applied. The larger the boundary strength is, the stronger filtering is applied. When it has been determined that the deblocking filter should be applied to a boundary, the deblocking process calculates the gradient on each side of the boundary to make a decision of how many samples should be modified on the respective side. The decision is denoted as either a “strong”, “weak” or “long tap” deblocking filter decision. For example, the VVC strong/weak filter decision is illustrated in the table below and is checked for line 0 and line 3 when filtering the horizontal border between two blocks above/below each other.

TABLE 8

If $2 \cdot d_0$ is less than $(\beta \gg 2)$, $Abs(p_{3,0} - p_{0,0}) + Abs(q_{0,0} - q_{3,0})$ is less than $(\beta \gg 3)$ and $Abs(p_{0,0} - q_{0,0})$ is less than $(5 \cdot tC + 1) \gg 1$

&&

$2*d_3$ is less than $(\beta \gg 2)$, $Abs(p_{3,3} - p_{0,3}) + Abs(q_{0,3} - q_{3,3})$ is less than $(\beta \gg 3)$ and $Abs(p_{0,3} - q_{0,3})$ is less than $(5 * tC + 1) \gg 1$

where $d_0 = abs(p_{0,0} - 2*p_{1,0} + p_{2,0}) + abs(q_{0,0} - 2*q_{1,0} + q_{2,0})$ and

$d_1 = abs(p_{0,3} - 2*p_{1,3} + p_{2,3}) + abs(q_{0,3} - 2*q_{1,3} + q_{2,3})$

If the strong/weak decision is true the strong filter is applied for line 0 to line 4 else the weak filter is applied.

Strong filter:

$p0' = Clip3(p0 - 2 * tC, p0 + 2 * tC, (p2 + 2 * p1 + 2 * p0 + 2 * q0 + q1 + 4) \gg 3)$

$p1' = Clip3(p1 - 2 * tC, p1 + 2 * tC, (p2 + p1 + p0 + q0 + 2) \gg 2)$

$p2' = Clip3(p2 - 2 * tC, p2 + 2 * tC, (2 * p3 + 3 * p2 + p1 + p0 + q0 + 4) \gg 3)$

$q0' = Clip3(q0 - 2 * tC, q0 + 2 * tC, (p1 + 2 * p0 + 2 * q0 + 2 * q1 + q2 + 4) \gg 3)$

$q1' = Clip3(q1 - 2 * tC, q1 + 2 * tC, (p0 + q0 + q1 + q2 + 2) \gg 2)$

$q2' = Clip3(q2 - 2 * tC, q2 + 2 * tC, (p0 + q0 + q1 + 3 * q2 + 2 * q3 + 4) \gg 3)$

Here p^ belongs to a line of block P and $p0$ is closest to the block boundary to block Q and where q^* belongs to a line of block Q and $q0$ is closest to block boundary to block P.*

$p3 \ p2 \ p1 \ p0 \mid q0 \ q1 \ q2 \ q3$

Weak filter:

$\Delta = (9 * (q0 - p0) - 3 * (q1 - p1) + 8) \gg 4$

When $Abs(\Delta)$ is less than $tC * 10$, the following ordered steps apply:

The filtered sample values $p0'$ and $q0'$ are specified as follows:

$\Delta = Clip3(-tC, tC, \Delta)$

$$p0' = \text{Clip1Y}(p0 + \Delta)$$

$$q0' = \text{Clip1Y}(q0 - \Delta)$$

The variables dp and dq are derived as follows:

$$dp = \text{Abs}(p2 - 2 * p1 + p0)$$

$$dq = \text{Abs}(q2 - 2 * q1 + q0)$$

When dp is less than $(\beta + (\beta \gg 1)) \gg 3$, the variable dEp is set equal to 1.

When dq is less than $(\beta + (\beta \gg 1)) \gg 3$, the variable dEq is set equal to 1.

When dEp is equal to 1, the filtered sample value p1' is specified as follows:

$$\Delta p = \text{Clip3}(-(tc \gg 1), tc \gg 1, (((p2 + p0 + 1) \gg 1) - p1 + \Delta) \gg 1)$$

$$p1' = \text{Clip1Y}(p1 + \Delta p)$$

When dEq is equal to 1, the filtered sample value q1' is specified as follows:

$$\Delta q = \text{Clip3}(-(tc \gg 1), tc \gg 1, (((q2 + q0 + 1) \gg 1) - q1 - \Delta) \gg 1)$$

$$q1' = \text{Clip1Y}(q1 + \Delta q)$$

[0033] The deblocking filter strength parameter is determined from a specified table based on the quantization parameter Q as table index. The quantization parameter Q is derived as: $Q = \text{Clip3}(0, 63, qP + (\text{slice_beta_offset_div2} \ll 1))$, where slice_beta_offset_div2 is the value of the syntax element slice_beta_offset_div2 for the slice that contains sample q0,0.

[0034] The deblocking filter strength parameter is determined from a specified table based on the quantization parameter Q as table index.

[0035] $Q = \text{Clip3}(0, 65, qP + 2 * (bS - 1) + (\text{slice_tc_offset_div2} \ll 1))$, where slice_tc_offset_div2 is the value of the syntax element slice_tc_offset_div2 for the slice that contains sample q0,0.

[0036] Furthermore, qP specifies the average qp of the coding units which include the coding blocks containing the sample $q0,0$ and $p0,0$. bS specifies the boundary filtering strength.

[0037] The Derivation of threshold variables β' and tC' from input Q is shown in the table below.

TABLE 9 – Derivation of threshold variables β' and tC' from input Q

Q	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
β'	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6
tC'	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Q	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
β'	7	8	9	10	11	12	13	14	15	16	17	18	20	22	24	26	28
tC'	0	3	4	4	4	4	5	5	5	5	7	7	8	9	10	10	11
Q	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
β'	30	32	34	36	38	40	42	44	46	48	50	52	54	56	58	60	62
tC'	13	14	15	17	19	21	24	25	29	33	36	41	45	51	57	64	71
Q	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65		
β'	64	66	68	70	72	74	76	78	80	82	84	86	88	-	-		
tC'	80	89	100	112	125	141	157	177	198	222	250	280	314	352	395		

[0038] In HEVC, the deblocking filter parameter offsets for β and tC that are applied to the Cb and Cr components are not signaled in the bitstream but set to be equal to the offsets that are applied to the luma component.

[0039] In VVC, the deblocking filter parameter offsets for β and tC that are applied to the Y, Cb and Cr components are separately signaled in the bitstream. As shown in the above Table, the threshold variables β' and tC' that are applied to the Cb and Cr components are derived by using input Q as index.

[0040] Here, Q that is applied to the Cb component is derived as below from β and tC respectively:

$$Q = \text{Clip3}(0, 63, qP + (\text{slice_cb_beta_offset_div2} \ll 1))$$

$$Q = \text{Clip3}(0, 65, qP + 2 * (bS - 1) + (\text{slice_cb_tc_offset_div2} \ll 1)).$$

[0041] Furthermore, Q that is applied to the Cr component is derived as below from β and tC respectively:

$$Q = \text{Clip3}(0, 63, qP + (\text{slice_cr_beta_offset_div2} \ll 1))$$

$$Q = \text{Clip3}(0, 65, qP + 2 * (bS - 1) + (\text{slice_cr_tc_offset_div2} \ll 1)).$$

SUMMARY

[0042] In this document, we use the syntax elements in the picture parameter set (PPS), picture header (PH) and slice header (SH) as examples. Those syntax elements include: 1) deblocking parameter offsets for β and tC (divided by 2) that are applied to the three color components Y, Cb and Cr (these syntax elements are called “deblocking offsets”); 2) deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb and Cr color components (these syntax elements are called “chroma deblocking offsets”); and 3) deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component (these syntax elements are called “luma deblocking offsets”).

[0043] 1. *General syntax construct problem*

[0044] In the current version of VVC, when a first syntax element (referred to as Syntax_A) is not signaled, the value of Syntax_A may be inferred to be equal to a first value. The first value may be either a constant value or the value of a second syntax element (referred to as Syntax_B). In other words, when Syntax_A is not signaled, the value of Syntax_A may not be inferred to be equal to the value of a third syntax element (referred to as Syntax_C).

[0045] 2. Problems of signaling deblocking related syntax elements

[0046] In the current version of VVC, the deblocking filter can be enabled / disabled by the syntax element `pps_deblocking_filter_disable_flag`, `ph_deblocking_filter_disabled_flag` and `slice_deblocking_filter_disabled_flag` in the PPS, PH and SH, respectively. When one of the flag is signaled and the value is equal to 0, the deblocking offsets are signaled in the PPS, PH and SH, respectively. When the deblocking offsets are not signaled in the stream, the values of the deblocking offsets are inferred as shown in the table below.

TABLE 10 The value inference of the deblocking offsets

	Not present in PPS	Not present in PH	Not present in SH
beta_offset_div2	pps_beta_offset_div2 = 0	ph_beta_offset_div2 = pps_beta_offset_div2	slice_beta_offset_div2 = ph_beta_offset_div2
tc_offset_div2	pps_tc_offset_div2 = 0	ph_tc_offset_div2 = pps_tc_offset_div2	slice_tc_offset_div2 = ph_tc_offset_div2
cb_beta_offset_div2	pps_cb_beta_offset_div2 = 0	ph_cb_beta_offset_div2 = pps_cb_beta_offset_div2	slice_cb_beta_offset_div2 = ph_cb_beta_offset_div2
cb_tc_offset_div2	pps_cb_tc_offset_div2 = 0	ph_cb_tc_offset_div2 = pps_cb_tc_offset_div2	slice_cb_tc_offset_div2 = ph_cb_tc_offset_div2
cr_beta_offset_div2	pps_cr_beta_offset_div2 = 0	ph_cr_beta_offset_div2 = pps_cr_beta_offset_div2	slice_cr_beta_offset_div2 = ph_cr_beta_offset_div2
cr_tc_offset_div2	pps_cr_tc_offset_div2 = 0	ph_cr_tc_offset_div2 = pps_cr_tc_offset_div2	slice_cr_tc_offset_div2 = ph_cr_tc_offset_div2

[0047] Problem one:

[0048] In the current version of VVC, when pps_deblocking_filter_disable_flag is present in the PPS and the value is equal to 0, the deblocking offsets are signaled in the PPS. However, when a video source that is represented by the bitstream only comprises luma samples (monochrome video) or is coded using three separate color planes (separate_colour_plane_flag is equal to 1), the deblocking filter is not applied to the chroma components (Cb and Cr). A problem with the current version of VVC is that the chroma deblocking offsets in some cases have to be signaled in the PPS even though they will not be used in the decoding process which is an unnecessary bit-rate cost.

[0049] Problem two:

[0050] In the current version of VVC, the deblocking offsets are signaled separately for the three color components to provide the flexibility of controlling the deblocking strength separately for each of the color components. However, one frequent deblocking filter setting uses the same deblocking strength for all color components. For instance, this has been the case for the common test conditions (CTC) that is used in the development of VVC, where the chroma deblocking offsets have the same values as the luma deblocking offsets. Signaling chroma deblocking offsets separately comes at an unnecessary bit-rate cost for this setting. Given that the deblocking offsets are signed integer 0-th order Exp-Golomb-coded syntax elements with range [-12, 12], the bit-rate cost becomes as shown in the table below:

TABLE 11. Bit cost for each of the deblocking offsets

Bit cost	Value of the syntax elements
1	{ 0 }
3	{ -1, 1 }
5	{ -3, -2, 2, 3 }
7	{ -7, -6, -5, -4, 4, 5, 6, 7 }
9	{ -12, -11, -10, -9, -8, 8, 9, 10, 11, 12 }

[0051] When deblocking offsets are signaled in the PPS, PH or SH, the bit cost of chroma deblocking offsets is at least 4 bits and at most 36 bits. The bit cost is here calculated as the bit cost for the deblocking offset value shown in the above table times four, which is the number of chroma deblocking offsets syntax elements.

[0052] Problem three:

[0053] In the current version of VVC, when the deblocking filter is enabled, it implies that the deblocking filter is enabled on all color components, e.g. Y, Cb and Cr. It is not possible to enable the deblocking filter on one or more color components and disable the deblocking filter for the remaining color components. For example, it is not possible to enable the deblocking filter on the luma component but disable the deblocking filter on the chroma components (Cb and Cr).

[0054] This disclosure proposes solutions to the above described problems.

[0055] According to a first aspect of the present disclosure, there is provided a method for deriving a value for a first syntax element, Syntax_A. The method comprises determining whether Syntax_A is present in a bitstream. The method comprises, as a result of determining that Syntax_A is not present in the bitstream, deriving the value for Syntax_A to be equal to a first value, B, if a first condition is satisfied, or deriving the value for Syntax_A to be equal to a second value, C, if a second condition is satisfied.

[0056] According to a second aspect of the present disclosure, there is provided a computer program comprising instructions which, when executed by processing circuitry, causes the processing circuitry to perform the method according to the first aspect.

[0057] According to a third aspect of the present disclosure, there is provided a carrier comprising the computer program according to the second aspect, wherein the carrier is one of an electronic signal, an optical signal, a radio signal, and a computer readable storage medium.

[0058] According to a fourth aspect of the present disclosure, there is provided an apparatus, the apparatus being adapted to perform the method according to the first aspect.

[0059] An advantage of the embodiments is to reduce bit-cost by inferring the value of a syntax element using the proposed inference method. Another advantage is to reduce the operation cost of deblocking filter in the decoder side by a deblocking filter enable / disable flag. Another advantage is to increase the flexibility of the deblocking filter control.

BRIEF DESCRIPTION OF THE DRAWINGS

[0060] The accompanying drawings, which are incorporated herein and form part of the specification, illustrate various embodiments.

[0061] FIG. 1 illustrates an example of dividing a CTU.

[0062] FIG. 2 illustrates a system according to an example embodiment.

[0063] FIG. 3 is a schematic block diagram of encoder according to an embodiment.

[0064] FIG. 4 is a schematic block diagram of encoder according to an embodiment.

[0065] FIG. 5 is a flowchart illustrating a process according to an embodiment.

[0066] FIG. 6 is a flowchart illustrating a process according to an embodiment.

[0067] FIG. 7 is a flowchart illustrating a process according to an embodiment.

[0068] FIG. 8 is a flowchart illustrating a process according to an embodiment.

[0069] FIG. 9 is a flowchart illustrating a process according to an embodiment.

[0070] FIG. 10 is a flowchart illustrating a process according to an embodiment.

[0071] FIG. 11 is a flowchart illustrating a process according to an embodiment.

[0072] FIG. 12 is a block diagram of an apparatus according to an embodiment.

DETAILED DESCRIPTION

[0073] FIG. 2 illustrates a system 200 according to an example embodiment. System 200 includes an encoder 202 in communication with a decoder 204 via a network 210 (e.g., the Internet or other network).

[0074] FIG. 3 is a schematic block diagram of encoder 202 for encoding a block of pixel values (hereafter “block”) in a video frame (picture) of a video sequence according to an embodiment. A current block is predicted by performing a motion estimation by a motion estimator 350 from an already provided block in the same frame or in a previous frame. The result of the motion estimation is a motion or displacement vector associated with the reference block, in the case of inter prediction. The motion vector is utilized by a motion compensator 350 for outputting an inter prediction of the block. An intra predictor 349 computes an intra prediction of the current block. The outputs from the motion estimator/compensator 350 and the intra predictor 349 are input in a selector 351 that either selects intra prediction or inter prediction for the current block. The output from the selector 351 is input to an error calculator in the form of an adder 341 that also receives the pixel values of the current block. The adder 341 calculates and outputs a residual error as the difference in pixel values between the block and its prediction. The error is transformed in a transformer 342, such as by a discrete cosine transform, and quantized by a quantizer 343 followed by coding in an encoder 344, such as by entropy encoder. In inter coding, also the estimated motion vector is brought to the encoder 344 for generating the coded representation of the current block. The transformed and quantized residual error for the current block is also provided to an inverse quantizer 345 and inverse transformer 346 to retrieve the original residual error. This error is added by an adder 347 to the block prediction output from the motion compensator 350 or the intra predictor 349 to create a reference block that can be used in the prediction and coding of a next block. This new reference block is first processed by a deblocking filter unit 330 according to the embodiments in order to perform deblocking filtering to combat any blocking artifact. The processed new reference block is then temporarily stored in a frame buffer 348, where it is available to the intra predictor 349 and the motion estimator/compensator 350.

[0075] FIG. 4 is a corresponding schematic block diagram of decoder 204 according to some embodiments. The decoder 204 comprises a decoder 461, such as entropy decoder, for decoding

an encoded representation of a block to get a set of quantized and transformed residual errors. These residual errors are dequantized in an inverse quantizer 462 and inverse transformed by an inverse transformer 463 to get a set of residual errors. These residual errors are added in an adder 464 to the pixel values of a reference block. The reference block is determined by a motion estimator/compensator 467 or intra predictor 466, depending on whether inter or intra prediction is performed. A selector 468 is thereby interconnected to the adder 464 and the motion estimator/compensator 467 and the intra predictor 466. The resulting decoded block output from the adder 464 is input to a deblocking filter unit 330 according to the embodiments in order to deblocking filter any blocking artifacts. The filtered block is output from the decoder 504 and is furthermore preferably temporarily provided to a frame buffer 465 and can be used as a reference block for a subsequent block to be decoded. The frame buffer 465 is thereby connected to the motion estimator/compensator 467 to make the stored blocks of pixels available to the motion estimator/compensator 467. The output from the adder 464 is preferably also input to the intra predictor 466 to be used as an unfiltered reference block.

[0076] Embodiments

[0077] In the description below, various embodiments are described that solve one or more of the above described problems. It is to be understood by a person skilled in the art that two or more embodiments, or parts of embodiments, may be combined to form new solutions which are still covered by this disclosure.

[0078] Embodiment 1 – A method for inferring a syntax element value

[0079] In the first embodiment, a method is proposed for determining a value for a syntax element Syntax_A. When Syntax_A is not present, depending on a first condition, the value of Syntax_A is inferred to be equal to a value B, where the value B is derived from at least a value for a syntax element Syntax_B, and depending on a second condition, the value of Syntax_A is inferred to be a value C, where the value C is derived from at least a value for a syntax element Syntax_C. In other words, when Syntax_A is not present, the value of Syntax_A can be inferred from the value of at least two different syntax elements.

[0080] Accordingly, in one embodiment the method includes the following steps.

[0081] Determining a value of a first syntax element (namely Syntax_C). Determining a value of a second syntax element (namely Syntax_B). Determining a value of a third syntax element (namely Syntax_D). Determining whether a fourth syntax element (namely Syntax_A) is present or not in the bitstream.

[0082] When it is determined that Syntax_A is not present in the bitstream, then: if the determined value of Syntax_D is equal to a first value, the value of Syntax_A is inferred to be a value B, where the value B is derived from at least a determined value of Syntax_B, and if the determined value of Syntax_D is equal to a second value, the value of Syntax_A is inferred to be a value C, where the value C is derived from at least a value of Syntax_C.

[0083] In one embodiment, determining the value of Syntax_B and Syntax_C comprises: determining whether the syntax element is present or not in the bitstream; and, if the syntax element is determined not present in the bitstream, the value of the syntax element is inferred, otherwise, the value of the syntax element is decoded.

[0084] In one embodiment, determining the value of Syntax_D comprises determining whether the syntax element present or not in the bitstream. If Syntax_D is determined present in the bitstream, the determined value of Syntax_D is a first value, where, the first value is a condition (e.g. Syntax_D is present) or the first value is equal to the decoded value of Syntax_D (e.g. 0). If Syntax_D is not present in the bitstream, then the determined value of Syntax_D is a second value, where, the second value is a condition (e.g. Syntax_D is not present) or the second value is equal to be an inferred value (e.g. 2).

[0085] The syntax elements Syntax_A, Syntax_B, Syntax_C, and Syntax_D may be syntax elements in one or several header or parameter sets.

[0086] FIG. 5 is a flowchart illustrating a process 500 for deriving a value for a syntax element (Syntax_A) according to the above description. Process 500 may be performed by a decoder.

[0087] Step s502 comprises deriving a value (D). In one embodiment, deriving D comprises: i) setting D=X (e.g., 1) if a certain syntax element (Syntax_D) is present in the bitstream and Syntax_D indicates that Syntax_A is included in the bistream, ii) setting D=Y (e.g., 0) if Syntax_D is present in the bitstream and Syntax_D indicates that Syntax_A is not included in the

bistream, or iii) otherwise setting D such that D does not equal either X or Y (e.g., setting D equal to Z).

[0088] Step s504 comprises determining whether D equals X. If D equals X, the process goes to step s506, otherwise the process goes to step s508.

[0089] Step s506 comprises setting the value for Syntax_A equal to a value A, where the value A is the decoded value of Syntax_A.

[0090] Step s508 comprises determining whether D equals Y. If D equals Y, the process goes to step s510, otherwise the process goes to step s512.

[0091] Step s510 comprises setting the value for Syntax_A equal to a value B, where the value B is a value associated with Syntax_B (e.g., B may be equal to the decoded value of Syntax_B).

[0092] Step s512 comprises setting the value for Syntax_A equal to a value C, where the value C is a value associated with Syntax_C (e.g., C may be equal to the decoded value of Syntax_C).

[0093] FIG. 6 is a flowchart illustrating a process 600 for deriving a value for a syntax element (Syntax_A) (a.k.a., the “sixth” syntax element). Process 600 may be performed by a decoder.

[0094] Step s602 comprises determining whether a first syntax element Syntax_C is present in the bitstream or not. If Syntax_C is present, the process goes to step s604, otherwise it goes to step s606.

[0095] Step s604 (Syntax_C present) comprises decoding the first syntax element Syntax_C and deriving a first value as equal to the value of the decoded first syntax element Syntax_C.

[0096] Step s606 (Syntax_C not present) comprises deriving the first value as equal to a second value, wherein, the second value may be a constant value, or the second value may be the value of a second syntax element.

[0097] Step s608 comprises determining whether a third syntax element Syntax_B is present in the bitstream or not. If Syntax_B is present, the process goes to step s610, otherwise it goes to step s612.

[0098] Step s610 (Syntax_B present) comprises decoding the third syntax element Syntax_B and deriving a third value as equal to the value of the decoded third syntax element Syntax_B.

[0099] Step s612 (Syntax_B not present) comprises deriving the third value as equal to a fourth value, wherein, the fourth value may be a constant value, or the fourth value may be the value of a fourth syntax element.

[00100] Step s614 comprises determining whether a fifth syntax element Syntax_D is present in the bitstream or not. If Syntax_D is present, the process goes to step s616, otherwise it goes to step s624.

[00101] Step s616 (Syntax_D present) comprises decoding the fifth syntax element Syntax_D and deriving a fifth value as equal to the value of the decoded fifth syntax element Syntax_D, wherein, the fifth value indicates whether a sixth syntax element Syntax_A is present in the bitstream or not.

[00102] Step s618 comprises determining, based on the fifth value, whether or not Syntax_A is present in the bitstream or not. If Syntax_A is present, the process goes to step s620, otherwise it goes to step s622.

[00103] Steps s620 (Syntax_A is present) comprises decoding the sixth syntax element Syntax_A and deriving a sixth value as equal to the value of the decoded sixth syntax element Syntax_A.

[00104] Step s622 (Syntax_A not present) comprises deriving the sixth value as equal to the third value, wherein, the third value is the value of the third syntax element Syntax_B.

[00105] Step s624 (Syntax_D not present) comprises deriving the sixth value as equal to the first value, wherein, the first value is the value of the first syntax element Syntax_C.

[00106] Step s626 comprises using the sixth value in a process to decode a picture wherein the Syntax_A, Syntax_B, Syntax_C, Syntax_D are four different syntax elements.

[00107] FIG. 7 is a flowchart illustrating a process 700 for inferring a value for a syntax element (Syntax_A). Process 700 may be performed by a decoder.

[00108] Step s702 comprises determining whether a first syntax element Syntax_C is present in the bitstream or not. If Syntax_C is present, the process goes to step s704, otherwise it goes to step s706.

[00109] Step s704 (Syntax_C present) comprises decoding the first syntax element Syntax_C and deriving a first value as equal to the value of the decoded first syntax element Syntax_C.

[00110] Step s706 (Syntax_C not present) comprises deriving the first value as equal to a second value, wherein, the second value may be a constant value, or the second value may be the value of a second syntax element.

[00111] Step s708 comprises determining whether a third syntax element Syntax_B is present in the bitstream or not. If Syntax_B is present, the process goes to step s710, otherwise it goes to step s712.

[00112] Step s710 (Syntax_B present) comprises decoding the third syntax element Syntax_B and deriving a third value as equal to the value of the decoded third syntax element Syntax_B.

[00113] Step s712 (Syntax_B not present) comprises deriving the third value as equal to a fourth value, wherein, the fourth value may be a constant value, or the fourth value may be the value of a fourth syntax element.

[00114] Step s714 comprises determining whether a fifth syntax element Syntax_D is present in the bitstream or not. If Syntax_D is present, the process goes to step s716, otherwise it goes to step s718.

[00115] Step s716 (Syntax_D present) comprises decoding the fifth syntax element Syntax_D and deriving a fifth value as equal to the value of the decoded fifth syntax element Syntax_D, wherein, the fifth value indicates whether a sixth syntax element Syntax_A is present in the bitstream or not.

[00116] Step s718 (Syntax_D not present) comprises deriving the fifth value as equal to a sixth value, wherein, the sixth value may be a constant value, or the sixth value may be the value of a sixth syntax element.

[00117] Step s702 comprises determining, based on the fifth value, whether a seventh syntax element Syntax_A is present in the bitstream or not (e.g., if the fifth value is equal to a value of

X (e.g., X=1), then this specifies that Syntax_A is present in the bitstream. If Syntax_A is present, the process goes to step s722, otherwise it goes to step s724.

[00118] Step s722 (Syntax_A is present) comprises decoding the seventh syntax element Syntax_A and deriving a seventh value as equal to the value of the decoded seventh syntax element Syntax_A.

[00119] Step s724 (Syntax_A is not present) comprises determining whether the fifth value is equal to a ninth value (denoted Y). If the fifth value equals Y, then the value of the seventh syntax element Syntax_A should be inferred as equal to the value of the third syntax element Syntax_B. If the fifth value equals Y, then the process goes to step s726, otherwise it goes to step s728.

[00120] Step s726 comprises deriving the seventh value as equal to the third value, wherein, the third value is the value of the third syntax element Syntax_B.

[00121] Step s728 comprises deriving the seventh value as equal to the first value, wherein, the first value is the value of the first syntax element Syntax_C.

[00122] Step s730 comprises using the seventh value in a process to decode the picture wherein the Syntax_A, Syntax_B, Syntax_C, Syntax_D are four different syntax elements.

[00123] Use Case 1 (FIG. 8) Inferring the value of chroma deblocking offsets in the PH

[00124] In this use case, a presence flag (which corresponds to Syntax_D described above) is added to the PH indicating whether the chroma deblocking offsets (which correspond to Syntax_A) are present in the PH or not. The flag may for instance be called

ph_deblocking_chroma_offsets_present_flag. When the proposed flag is equal to a first value (e.g. 1), it is determined that the chroma deblocking offsets are signaled in the bitstream. When the proposed flag is present and the value of the flag is equal to a second value (e.g. 0), it is determined that the chroma deblocking offsets are not signaled but set to be equal to the values of the luma deblocking offsets in the PH. When the proposed flag is not present, it is determined that the chroma deblocking offsets are not signaled but should be set to be equal to the values of chroma deblocking offsets in the PPS.

[00125] The proposed method in this uses case is based on embodiment 1 in the way that when Syntax_A is not present, and the determined value of Syntax_D is equal to a first value (e.g. the first value is that Syntax_D is present), the value of Syntax_A is inferred to be the value of Syntax_B, if the determined value of Syntax_D is equal to a second value (e.g. the second value is that Syntax_D is not present), the value of Syntax_A is inferred to be the value of Syntax_C, wherein the Syntax_A represents one of the four syntax elements: ph_cb_beta_offset_div2, ph_cb_tc_offset_div2, ph_cr_beta_offset_div2 or ph_cr_tc_offset_div2. Below we have set Syntax_A equal to the ph_cb_beta_offset_div2 syntax element as an example to show the syntax construct pseudo code:

```

if ( Syntax_D == 1 ) {
    // Syntax_D and Syntax_A are present in the bitstream
    Syntax_A = decoded value of Syntax_A in the bitstream
}
else // Syntax_A is not present in the bitstream
{
    if ( Syntax_D is present ) {
        Syntax_A = Syntax_B
    }
    else // Syntax_D is not present
    {
        Syntax_A = Syntax_C
    }
}
}
Where,
    Syntax_A = ph_cb_beta_offset_div2
    Syntax_B = ph_beta_offset_div2
    Syntax_C = pps_cb_beta_offset_div2
    Syntax_D = ph_deblocking_chroma_offsets_present_flag

```

[00126] In another version of this embodiment, at least one of Syntax_B, Syntax_C and Syntax_D is a constant value

[00127] In another version of this embodiment, at least one of Syntax_B, Syntax_C and Syntax_D is a value derived from one or more syntax elements in the bitstream.

[00128] Below are example syntax and semantics for use case 1.

TABLE 14

pic_parameter_set_rbsp() {	Descriptor
...	...
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
pps_deblocking_filter_disabled_flag OR pps_deblocking_filter_enabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag OR	
pps_deblocking_filter_enabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
...	...
rbp_trailing_bits()	
}	

[00129] **deblocking_filter_control_present_flag** equal to 1 specifies the presence of deblocking filter control syntax elements in the PPS. **deblocking_filter_control_present_flag** equal to 0 specifies the absence of deblocking filter control syntax elements in the PPS.

[00130] **deblocking_filter_override_enabled_flag** equal to 1 specifies the presence of **ph_deblocking_filter_override_flag** in the PHs referring to the PPS or **slice_deblocking_filter_override_flag** in the slice headers referring to the PPS. **deblocking_filter_override_enabled_flag** equal to 0 specifies the absence of **ph_deblocking_filter_override_flag** in PHs referring to the PPS or

slice_deblocking_filter_override_flag in slice headers referring to the PPS. When not present, the value of deblocking_filter_override_enabled_flag is inferred to be equal to 0.

[00131] pps_deblocking_filter_disabled_flag equal to 1 specifies that the operation of deblocking filter is not applied for slices referring to the PPS in which slice_deblocking_filter_disabled_flag is not present. pps_deblocking_filter_disabled_flag equal to 0 specifies that the operation of the deblocking filter is applied for slices referring to the PPS in which slice_deblocking_filter_disabled_flag is not present. When not present, the value of pps_deblocking_filter_disabled_flag is inferred to be equal to 0. In some embodiments, the pps_deblocking_filter_disabled_flag is replaced with the pps_deblocking_filter_enabled_flag, which is equivalent to !pps_deblocking_filter_disabled_flag. Accordingly, pps_deblocking_filter_enabled_flag equal to 0 specifies that the operation of deblocking filter is not applied for slices referring to the PPS in which slice_deblocking_filter_disabled_flag is not present; pps_deblocking_filter_enabled_flag equal to 1 specifies that the operation of the deblocking filter is applied for slices referring to the PPS in which slice_deblocking_filter_disabled_flag is not present. When not present, the value of pps_deblocking_filter_enabled_flag is inferred to be equal to 1.

[00132] pps_beta_offset_div2 and **pps_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps_beta_offset_div2 and pps_tc_offset_div2 shall both be in the range of -12 to 12 , inclusive. When not present, the values of pps_beta_offset_div2 and pps_tc_offset_div2 are both inferred to be equal to 0.

[00133] pps_cb_beta_offset_div2 and **pps_cb_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 shall both be in the range of -12 to 12 , inclusive. When not present, the values of pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 are both inferred to be equal to 0.

[00134] **pps_cr_beta_offset_div2** and **pps_cr_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of **pps_cr_beta_offset_div2** and **pps_cr_tc_offset_div2** shall both be in the range of -12 to 12, inclusive. When not present, the values of **pps_cr_beta_offset_div2** and **pps_cr_tc_offset_div2** are both inferred to be equal to 0.

TABLE 15

picture_header_structure() {	Descriptor
...	...
if(deblocking_filter_override_enabled_flag && dbf_info_in_ph_flag) {	
ph_deblocking_filter_override_flag	u(1)
if(ph_deblocking_filter_override_flag) {	
ph_deblocking_filter_disabled_flag OR ph_deblocking_filter_enabled_flag	u(1)
if(!ph_deblocking_filter_disabled_flag OR ph_deblocking_filter_enabled_flag)	
{	
ph_beta_offset_div2	se(v)
ph_tc_offset_div2	se(v)
if(ChromaArrayType != 0) {	
ph_deblocking_chroma_offsets_present_flag	u(1)
if(ph_deblocking_chroma_offsets_present_flag) {	
ph_cb_beta_offset_div2	se(v)
ph_cb_tc_offset_div2	se(v)
ph_cr_beta_offset_div2	se(v)
ph_cr_tc_offset_div2	se(v)
}	
}	
}	
}	

[00135] **ph_deblocking_filter_override_flag** equal to 1 specifies that deblocking parameters are present in the PH. **ph_deblocking_filter_override_flag** equal to 0 specifies that deblocking

parameters are not present in the PH. When not present, the value of `ph_deblocking_filter_override_flag` is inferred to be equal to 0.

[00136] `ph_deblocking_filter_disabled_flag` equal to 1 specifies that the operation of the deblocking filter is not applied for the slices associated with the PH.

`ph_deblocking_filter_disabled_flag` equal to 0 specifies that the operation of the deblocking filter is applied for the slices associated with the PH. When `ph_deblocking_filter_disabled_flag` is not present, it is inferred to be equal to `pps_deblocking_filter_disabled_flag`. In some embodiments, the `ph_deblocking_filter_disabled_flag` is replaced with the `ph_deblocking_filter_enabled_flag` which is equivalent to `!ph_deblocking_filter_disabled_flag`. Accordingly, `ph_deblocking_filter_enabled_flag` equal to 0 specifies that the operation of the deblocking filter is not applied for the slices associated with the PH; `ph_deblocking_filter_enabled_flag` equal to 1 specifies that the operation of the deblocking filter is applied for the slices associated with the PH. When `ph_deblocking_filter_enabled_flag` is not present, it is inferred to be equal to `!pps_deblocking_filter_disabled_flag` or it is inferred to be equal to `pps_deblocking_filter_enabled_flag`.

[00137] `ph_beta_offset_div2` and `ph_tc_offset_div2` specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for the slices associated with the PH. The values of `ph_beta_offset_div2` and `ph_tc_offset_div2` shall both be in the range of -12 to 12 , inclusive. When not present, the values of `ph_beta_offset_div2` and `ph_tc_offset_div2` are inferred to be equal to `pps_beta_offset_div2` and `pps_tc_offset_div2`, respectively.

[00138] `ph_deblocking_chroma_offsets_present_flag` equal to 1 specifies the presence of deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb and Cr components in the PH. `ph_deblocking_chroma_offsets_present_flag` equal to 0 specifies the absence of deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb and Cr components in the PH. When not present, the value of `ph_deblocking_chroma_offsets_present_flag` is inferred to be equal to 0.

[00139] `ph_cb_beta_offset_div2` and `ph_cb_tc_offset_div2` specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for the slices

associated with the PH. The values of `ph_cb_beta_offset_div2` and `ph_cb_tc_offset_div2` shall both be in the range of -12 to 12 , inclusive. When not present, the values of `ph_cb_beta_offset_div2` and `ph_cb_tc_offset_div2` are inferred as follows: 1) when `ph_deblocking_chroma_offsets_present_flag` is present, the values of `ph_cb_beta_offset_div2` and `ph_cb_tc_offset_div2` are inferred to be equal to `ph_beta_offset_div2` and `ph_tc_offset_div2`, respectively; and 2) when `ph_deblocking_chroma_offsets_present_flag` is not present, the values of `ph_cb_beta_offset_div2` and `ph_cb_tc_offset_div2` are inferred to be equal to `pps_cb_beta_offset_div2` and `pps_cb_tc_offset_div2`, respectively.

[00140] `ph_cr_beta_offset_div2` and `ph_cr_tc_offset_div2` specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for the slices associated with the PH. The values of `ph_cr_beta_offset_div2` and `ph_cr_tc_offset_div2` shall both be in the range of -12 to 12 , inclusive. When not present, the values of `ph_cr_beta_offset_div2` and `ph_cr_tc_offset_div2` are inferred as follows: 1) when `ph_deblocking_chroma_offsets_present_flag` is present, the values of `ph_cr_beta_offset_div2` and `ph_cr_tc_offset_div2` are inferred to be equal to `ph_beta_offset_div2` and `ph_tc_offset_div2`, respectively; and 2) when `ph_deblocking_chroma_offsets_present_flag` is not present, the values of `ph_cr_beta_offset_div2` and `ph_cr_tc_offset_div2` are inferred to be equal to `pps_cr_beta_offset_div2` and `pps_cr_tc_offset_div2`, respectively.

[00141] The construct can be explained by the following decoding flowchart shown in FIG. 8 referring to the syntax table above.

[00142] Step s802 comprises determining the value of `pps_cb_beta_offset_div2`.

[00143] Step s804 comprises determining whether `ph_beta_offset_div2` is present in the bitstream. If it is process 800 goes to step s806, otherwise it goes to step s818. In an alternative embodiment, step s804 comprises determining whether a certain flag (e.g., a `ph_deblocking_filter_disabled_flag` or a `ph_deblocking_filter_enabled_flag`) is equal to $v1$ or $v2$, where $v1$ is a first value and $v2$ is a second value and $v2$ does not equal $v1$. When a certain flag is present and has a value equal to $v1$, process 800 goes to step s806, otherwise (the flag is either not present or present but has a value equal to $v2$), process 800 goes to step s818. In embodiments where the certain flag is the `ph_deblocking_filter_disabled_flag`, $v1=0$, and

v2=1. In embodiments where the certain flag is the `ph_deblocking_filter_enabled_flag`, v1=1, and v2=0.

[00144] Step s806 comprises decoding `ph_beta_offset_div2`.

[00145] Step s808 comprises determining if `ph_deblocking_chroma_offsets_present_flag` is present in the bitstream. If it is, process 800 goes to step s810, otherwise it goes to step s818. In an alternative embodiment, step s808 comprises determining whether `chromaArrayType` is equal to 0 or not. When `chromaArrayType` is equal to 0, process 800 goes to step s818, otherwise (`chromaArrayType` is not equal to 0), process 800 goes to s810.

[00146] Step s810 comprises decoding the `ph_deblocking_chroma_offsets_present_flag`.

[00147] Step s812 comprises determining whether `ph_deblocking_chroma_offsets_present_flag` is equal to 1. If it is, then process 800 goes to step s814, otherwise it goes to step s816.

[00148] Step s814 comprises decoding `ph_cb_beta_offset_div2`.

[00149] Step s816 comprises inferring `ph_cb_beta_offset_div2` as equal to the value of `ph_beta_offset_div2`.

[00150] Step s818 comprises inferring `ph_cb_beta_offset_div2` as equal to the value of `pps_cb_beta_offset_div2`.

[00151] Use Case 2 (FIG. 9): Inferring the value of chroma deblocking offsets in the SH

[00152] It is proposed to add a present flag to the SH indicating whether the chroma deblocking offsets are present in the SH or not. The flag may for instance be called **`slice_deblocking_chroma_offsets_present_flag`**. When the proposed flag is equal to a first value (e.g. 1), it is determined that the chroma deblocking offsets are signaled in the bitstream. When the proposed flag is present and the value of the flag equal to a second value (e.g. 0), it is determined that the chroma deblocking offsets are not signaled but set to be equal to the values of the luma deblocking offsets in the SH. When the proposed flag is not present, it is determined that the chroma deblocking offsets are not signaled but set to be equal to the values of chroma deblocking offsets in the PH.

[00153] The proposed method is based on embodiment 1 in the way that when Syntax_A is not present, and the determined value of Syntax_D is equal to a first value (e.g. the first value is that Syntax_D is present), the value of Syntax_A is inferred to be the value of Syntax_B, if the determined value of Syntax_D is equal to a second value (e.g. the second value is that Syntax_D is not present), the value of Syntax_A is inferred to be the value of Syntax_C, wherein the Syntax_A represents one of the four syntax elements: slice_cb_beta_offset_div2, slice_cb_tc_offset_div2, slice_cr_beta_offset_div2 or slice_cr_tc_offset_div2. Here we use Syntax_A represents slice_cb_beta_offset_div2 as an example to show the syntax construct pseudo code:

```

if ( Syntax_D == 1 )
{
    // Syntax_D and Syntax_A are present in the bitstream
    Syntax_A = decoded value of Syntax_A in the bitstream
}
else // Syntax_A is not present in the bitstream
{
    if ( Syntax_D is present)
    {
        Syntax_A = Syntax_B
    }
    else // Syntax_D is not present
    {
        Syntax_A = Syntax_C
    }
}
Where,
Syntax_A = slice_cb_beta_offset_div2
Syntax_B = slice_beta_offset_div2
Syntax_C = ph_cb_beta_offset_div2
Syntax_D = slice_deblocking_chroma_offsets_present_flag

```

[00154] In another version of this embodiment, at least one of Syntax_B, Syntax_C and Syntax_D is a constant value, or a value derived from one or more syntax elements in the bitstream. Below are example syntax and semantics for this use case.

TABLE 16

pic_parameter_set_rbsp() {	Descriptor
...	...

deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
pps_deblocking_filter_disabled_flag OR pps_deblocking_filter_enabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag OR pps_deblocking_filter_enabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
...	...
rbsp_trailing_bits()	
}	

[00155] deblocking_filter_control_present_flag equal to 1 specifies the presence of deblocking filter control syntax elements in the PPS. **deblocking_filter_control_present_flag** equal to 0 specifies the absence of deblocking filter control syntax elements in the PPS.

[00156] deblocking_filter_override_enabled_flag equal to 1 specifies the presence of **ph_deblocking_filter_override_flag** in the PHs referring to the PPS or **slice_deblocking_filter_override_flag** in the slice headers referring to the PPS. **deblocking_filter_override_enabled_flag** equal to 0 specifies the absence of **ph_deblocking_filter_override_flag** in PHs referring to the PPS or **slice_deblocking_filter_override_flag** in slice headers referring to the PPS. When not present, the value of **deblocking_filter_override_enabled_flag** is inferred to be equal to 0.

[00157] pps_deblocking_filter_disabled_flag equal to 1 specifies that the operation of deblocking filter is not applied for slices referring to the PPS in which **slice_deblocking_filter_disabled_flag** is not present. **pps_deblocking_filter_disabled_flag** equal to 0 specifies that the operation of the deblocking filter is applied for slices referring to the PPS in which **slice_deblocking_filter_disabled_flag** is not present. When not present, the value of **pps_deblocking_filter_disabled_flag** is inferred to be equal to 0. As noted above, the **pps_deblocking_filter_disabled_flag** may be replaced with the

pps_deblocking_filter_enabled_flag. As noted above, the pps_deblocking_filter_disabled_flag may be replaced with the pps_deblocking_filter_enabled_flag.

[00158] **pps_beta_offset_div2** and **pps_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps_beta_offset_div2 and pps_tc_offset_div2 shall both be in the range of -12 to 12 , inclusive. When not present, the values of pps_beta_offset_div2 and pps_tc_offset_div2 are both inferred to be equal to 0.

[00159] **pps_cb_beta_offset_div2** and **pps_cb_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 shall both be in the range of -12 to 12 , inclusive. When not present, the values of pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 are both inferred to be equal to 0.

[00160] **pps_cr_beta_offset_div2** and **pps_cr_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps_cr_beta_offset_div2 and pps_cr_tc_offset_div2 shall both be in the range of -12 to 12 , inclusive. When not present, the values of pps_cr_beta_offset_div2 and pps_cr_tc_offset_div2 are both inferred to be equal to 0.

TABLE 17

slice_header() {	Descriptor
...	...
if(deblocking_filter_override_enabled_flag && !dbf_info_in_ph_flag) {	
slice_deblocking_filter_override_flag	u(1)
if(slice_deblocking_filter_override_flag) {	
slice_deblocking_filter_disabled_flag OR slice_deblocking_filter_enabled_flag	u(1)

if(!slice_deblocking_filter_disabled_flag OR slice_deblocking_filter_enabled_flag) {	
slice_beta_offset_div2	se(v)
slice_tc_offset_div2	se(v)
if(ChromaArrayType != 0) {	
slice_deblocking_chroma_offsets_present_flag	u(1)
if(slice_deblocking_chroma_offsets_present_flag) {	
slice_cb_beta_offset_div2	se(v)
slice_cb_tc_offset_div2	se(v)
slice_cr_beta_offset_div2	se(v)
slice_cr_tc_offset_div2	se(v)
}	
}	
}	
}	
}	
}	
}	

[00161] **slice_deblocking_filter_override_flag** equal to 1 specifies that deblocking parameters are present in the slice header. **slice_deblocking_filter_override_flag** equal to 0 specifies that deblocking parameters are not present in the slice header. When not present, the value of **slice_deblocking_filter_override_flag** is inferred to be equal to **ph_deblocking_filter_override_flag**.

[00162] **slice_deblocking_filter_disabled_flag** equal to 1 specifies that the operation of the deblocking filter is not applied for the current slice. **slice_deblocking_filter_disabled_flag** equal to 0 specifies that the operation of the deblocking filter is applied for the current slice. When **slice_deblocking_filter_disabled_flag** is not present, it is inferred to be equal to **ph_deblocking_filter_disabled_flag**, or it is inferred to be equal to **!ph_deblocking_filter_enabled_flag**. The **slice_deblocking_filter_disabled_flag** may be replaced with the **slice_deblocking_filter_enabled_flag**, which is equivalent to **!slice_deblocking_filter_disabled_flag**.

[00163] **slice_beta_offset_div2** and **slice_tc_offset_div2** specify the deblocking parameter offsets for β and t_C (divided by 2) that are applied to the luma component for the current slice. The values of **slice_beta_offset_div2** and **slice_tc_offset_div2** shall both be in the range of -12

to 12, inclusive. When not present, the values of `slice_beta_offset_div2` and `slice_tc_offset_div2` are inferred to be equal to `ph_beta_offset_div2` and `ph_tc_offset_div2`, respectively.

[00164] `slice_deblocking_chroma_offsets_present_flag` equal to 1 specifies the presence of deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb and Cr components in the slice header. `slice_deblocking_chroma_offsets_present_flag` equal to 0 specifies the absence of deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb and Cr components in the slice header. When not present, the value of `slice_deblocking_chroma_offsets_present_flag` is inferred to be equal to 0.

[00165] `slice_cb_beta_offset_div2` and `slice_cb_tc_offset_div2` specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for the current slice. The values of `slice_cb_beta_offset_div2` and `slice_cb_tc_offset_div2` shall both be in the range of -12 to 12, inclusive. When not present, the values of `slice_cb_beta_offset_div2` and `slice_cb_tc_offset_div2` are inferred as follows: i) when `slice_deblocking_chroma_offsets_present_flag` is present, the value of `slice_cb_beta_offset_div2` and `slice_cb_tc_offset_div2` are inferred to be equal to `slice_beta_offset_div2` and `slice_tc_offset_div2`, respectively; and 2) when `slice_deblocking_chroma_offsets_present_flag` is not present, the value of `slice_cb_beta_offset_div2` and `slice_cb_tc_offset_div2` are inferred to be equal to `ph_cb_beta_offset_div2` and `ph_cb_tc_offset_div2`, respectively.

[00166] `slice_cr_beta_offset_div2` and `slice_cr_tc_offset_div2` specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for the current slice. The values of `slice_cr_beta_offset_div2` and `slice_cr_tc_offset_div2` shall both be in the range of -12 to 12, inclusive. When not present, the values of `slice_cr_beta_offset_div2` and `slice_cr_tc_offset_div2` are inferred as follows: i) When `slice_deblocking_chroma_offsets_present_flag` is present, the value of `slice_cr_beta_offset_div2` and `slice_cr_tc_offset_div2` are inferred to be equal to `slice_beta_offset_div2` and `slice_tc_offset_div2`, respectively; and 2) When `slice_deblocking_chroma_offsets_present_flag` is not present, the value of `slice_cr_beta_offset_div2` and `slice_cr_tc_offset_div2` are inferred to be equal to `ph_cr_beta_offset_div2` and `ph_cr_tc_offset_div2`, respectively.

[00167] The construct can be explained by the following decoding flowchart shown in FIG. 9 referring to the syntax table above.

[00168] Step s902 comprises determining the value of `ph_cb_beta_offset_div2`.

[00169] Step s904 comprises determining whether `slice_beta_offset_div2` is present in the bitstream. If it is process 900 goes to step s906, otherwise it goes to step s918. In an alternative embodiment, step s804 comprises determining whether a certain flag (e.g., a `slice_deblocking_filter_disabled_flag` or a `slice_deblocking_filter_enabled_flag`) is equal to `v1` or `v2`, where `v1` is a first value and `v2` is a second value and `v2` does not equal `v1`. When the certain flag is present and has a value equal to `v1` (e.g., 0), process 900 goes to step s906, otherwise (the flag is either not present or present but has a value equal to `v2`), process 900 goes to step s918. In embodiments where the certain flag is the `slice_deblocking_filter_disabled_flag`, `v1=0`, and `v2=1`. In embodiments where the certain flag is the `slice_deblocking_filter_enabled_flag`, `v1=1`, and `v2=0`.

[00170] Step s906 comprises decoding `slice_beta_offset_div2`.

[00171] Step s908 comprises determining if `slice_deblocking_chroma_offsets_present_flag` is present in the bitstream. If it is, process 900 goes to step s910, otherwise it goes to step s918. In an alternative embodiment, step s904 comprises determining whether `chromaArrayType` is equal to 0 or 1. When `chromaArrayType` is equal to 0, process 900 goes to step s918, otherwise (`chromaArrayType` is not equal to 0), process 900 goes to s910.

[00172] Step s910 comprises decoding the `slice_deblocking_chroma_offsets_present_flag`.

[00173] Step s912 comprises determining whether `slice_deblocking_chroma_offsets_present_flag` is equal to 1. If it is, then process 900 goes to step s914, otherwise it goes to step s916.

[00174] Step s914 comprises decoding `slice_cb_beta_offset_div2`.

[00175] Step s916 comprises inferring `slice_cb_beta_offset_div2` as equal to the value of `slice_beta_offset_div2`.

[00176] Step s918 comprises inferring `slice_cb_beta_offset_div2` as equal to the value of `ph_cb_beta_offset_div2`.

[00177] Use Case 3 – Inferring the value of chroma deblocking offsets in the PPS

[00178] It is proposed to add a present flag to the PPS indicating whether the chroma deblocking offsets are present in the PPS or not. The flag may for instance be called **pps_deblocking_chroma_offsets_present_flag**. When the proposed flag is equal to a first value (e.g. 1), it is determined that the chroma deblocking offsets are signaled in the bitstream. When the proposed flag is present and the value is equal to a second value (e.g. 0), it is determined that the chroma deblocking offsets are not signaled but set to be equal to the values of the luma deblocking offsets in the PPS. When the proposed flag is not present, it is determined that the chroma deblocking offsets are not signaled but set to be equal to 0.

[00179] Below are example syntax and semantics for this use case.

TABLE 18

pic_parameter_set_rbsp() {	Descriptor
...	...
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
pps_deblocking_filter_disabled_flag OR pps_deblocking_filter_enabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag OR pps_deblocking_filter_enabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
pps_deblocking_chroma_offsets_present_flag	u(1)
if(pps_deblocking_chroma_offsets_present_flag) {	
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
}	
...	...
rbsp_trailing_bits()	
}	

[00180] **deblocking_filter_control_present_flag** equal to 1 specifies the presence of deblocking filter control syntax elements in the PPS. **deblocking_filter_control_present_flag** equal to 0 specifies the absence of deblocking filter control syntax elements in the PPS.

[00181] **deblocking_filter_override_enabled_flag** equal to 1 specifies the presence of **ph_deblocking_filter_override_flag** in the PHs referring to the PPS or **slice_deblocking_filter_override_flag** in the slice headers referring to the PPS. **deblocking_filter_override_enabled_flag** equal to 0 specifies the absence of **ph_deblocking_filter_override_flag** in PHs referring to the PPS or **slice_deblocking_filter_override_flag** in slice headers referring to the PPS. When not present, the value of **deblocking_filter_override_enabled_flag** is inferred to be equal to 0.

[00182] **pps_deblocking_filter_disabled_flag** equal to 1 specifies that the operation of deblocking filter is not applied for slices referring to the PPS in which **slice_deblocking_filter_disabled_flag** is not present. **pps_deblocking_filter_disabled_flag** equal to 0 specifies that the operation of the deblocking filter is applied for slices referring to the PPS in which **slice_deblocking_filter_disabled_flag** is not present. When not present, the value of **pps_deblocking_filter_disabled_flag** is inferred to be equal to 0.

[00183] **pps_beta_offset_div2** and **pps_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of **pps_beta_offset_div2** and **pps_tc_offset_div2** shall both be in the range of -12 to 12 , inclusive. When not present, the values of **pps_beta_offset_div2** and **pps_tc_offset_div2** are both inferred to be equal to 0.

[00184] **pps_deblocking_chroma_offsets_present_flag** equal to 1 specifies the presence of deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb and Cr components in the PPS. **pps_deblocking_chroma_offsets_present_flag** equal to 0 specifies the absence of deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb and Cr components in the PPS. When not present, the value of **pps_deblocking_chroma_offsets_present_flag** is inferred to be equal to 0. When

ChromaArrayType is equal to 0, the value of pps_deblocking_chroma_offsets_present_flag shall be equal to 0.

[00185] pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 shall both be in the range of -12 to 12, inclusive. When not present, the values of pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 are inferred as follows: i) When pps_deblocking_chroma_offsets_present_flag is present, the values of pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 are inferred to be equal to pps_beta_offset_div2 and pps_tc_offset_div2, respectively; and 2) When pps_deblocking_chroma_offsets_present_flag is not present, the values of pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 are both inferred to be equal to 0.

[00186] pps_cr_beta_offset_div2 and pps_cr_tc_offset_div2 specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps_cr_beta_offset_div2 and pps_cr_tc_offset_div2 shall both be in the range of -12 to 12, inclusive. When not present, the values of pps_cr_beta_offset_div2 and pps_cr_tc_offset_div2 are inferred as follows: 1) When pps_deblocking_chroma_offsets_present_flag is present, the values of pps_cr_beta_offset_div2 and pps_cr_tc_offset_div2 are inferred to be equal to pps_beta_offset_div2 and pps_tc_offset_div2, respectively; and 2) When pps_deblocking_chroma_offsets_present_flag is not present, the values of pps_cr_beta_offset_div2 and pps_cr_tc_offset_div2 are both inferred to be equal to 0.

[00187] This embodiment discloses a novel syntax construct and method for inferring values when syntax elements are not present. A novel element here is the way the value of a syntax element that is not present in the bitstream is inferred. In short, if the syntax element is present, the value is derived as equal to the value of the decoded syntax element. But if the syntax

element is not present, the value is set equal to a value A or a value B depending on whether another syntax element is present in the bitstream or not.

[00188] Although the novel construct was above shown in a picture parameter set (PPS), this construct is not limited to the PPS. It may be used in e.g. a sequence parameter set, picture parameter set, slice header, video parameter set, decoding capability information, adaptation parameter set, supplemental enhancement information or picture header.

[00189] The construct can be explained by the following decoding method flowchart shown in FIG. 10 referring to the syntax table above. FIG. 10 illustrates a process 1000 according to an embodiment.

[00190] Step s1002 of process 1000 comprises determining whether luma deblocking offsets are present in the PPS or not. In one embodiment, determining whether luma deblocking offsets are present in the PPS or not comprises determining whether the value of `pps_deblocking_filter_disabled_flag` is equal to 0 or 1. Here 0 means that luma deblocking offsets are present, but the flag could alternatively be implemented in an opposite way where the value 1 means that they are present. If they are present, the process goes to step s1004, otherwise it goes to step s1014.

[00191] Step s1004 comprises decoding the luma deblocking offsets from the bitstream and a present flag is also decoded.

[00192] Step s1008 comprises determining whether the present flag specifies that chroma deblocking offsets are present. If the present flag specifies that chroma deblocking offsets are present, then the process goes to step s1010, otherwise it goes to step s1012.

[00193] Step s1010 (i.e., chroma deblocking offsets are present) comprises decoding the chroma deblocking offsets (e.g., decoding `pps_cb_beta_offset_div2`).

[00194] Step s1012 (i.e., chroma deblocking offsets not present) comprises inferring the values of the chroma deblocking offsets to be equal to the values of the luma deblocking offsets (e.g., inferring `pps_cb_beta_offset_div2` as equal to the value of `pps_beta_offset_div2`).

[00195] Step s1014 (i.e., the luma deblocking offsets are not present) comprises inferring the chroma deblocking offsets to be equal to 0.

[00196] Step s1016 comprises using the chroma deblocking offsets in a process to decode the picture.

[00197] This novel construct can be used for signaling of other values than deblocking filter parameters. A flow chart of the generalized construct is shown in FIG. 11, which is a flowchart illustrating a process 1100 according to an embodiment. A decoder may perform all or a subset of the following steps in this generalized embodiment to decode a picture from a bitstream.

[00198] Step s1102 comprises determining whether a first syntax element S1 is present in the bitstream or not. If S1 is in the bitstream, the process goes to step s1104, otherwise it goes to step s1112.

[00199] Step s1104 (i.e., S1 is determined to be present in the bitstream) comprises decoding the first syntax element S1 and deriving a first value as equal to the value of the decoded first syntax element S1.

[00200] Step s1106 comprises decoding a first present flag value from a second syntax element S2 in the bitstream wherein the first present flag value indicates whether a third syntax element S3 is present in the bitstream or not.

[00201] Step s1107 comprises, using the first present flag to determine whether S3 is present in the bitstream. If S3 is present, the process goes to step s1108, otherwise it goes to step s1110.

[00202] Step s1108 (i.e., the first present flag value indicates that a third syntax element S3 is present in the bitstream) comprises decoding the third syntax element S3 and deriving a third value as equal to the value of the decoded third syntax element S3.

[00203] Step s1110 (i.e., the first present flag value indicates that a third syntax element S3 is present in the bitstream) comprises deriving the third value as equal to a second value.

[00204] Step s1112 (i.e., S1 is determined to not be present in the bitstream) comprises deriving a third value as equal to a fourth value

[00205] Step s1114 comprises using the third value in a process to decode the picture wherein the first (S1), second (S2) and third (S3) syntax elements are three different syntax elements.

[00206] Use case 2 – An alternative way to indicate the presence and inference of the chroma deblocking offsets

[00207] In other uses cases described herein, the presence of the chroma deblocking offsets is conditional based on the values of **pps_deblocking_chroma_offsets_present_flag**, **ph_deblocking_chroma_offsets_present_flag** and **slice_deblocking_chroma_offsets_present_flag** in the PPS, PH and SH, respectively.

[00208] In this use case, an alternative way to indicate the presence and inference of the chroma deblocking offsets is proposed. It is proposed to add a syntax element to each of the PPS, PH and SH indicating whether the chroma deblocking offsets are present in the PPS, PH, SH or not present at all.

[00209] The syntax elements may be called **pps_deblocking_chroma_offsets_present_idc**, **ph_deblocking_chroma_offsets_present_idc** and **slice_deblocking_chroma_offsets_present_idc**. When a proposed syntax element is equal to a first value (e.g. 1), it is determined that the chroma deblocking offsets are signaled in the corresponding syntax structure (PPS, PH or SH). When a proposed syntax element is equal to a second value (e.g. 0), it is determined the chroma deblocking offsets are not signaled but set equal to the values of the luma deblocking offsets of the corresponding syntax structure (PPS, PH or SH). When a proposed syntax element is equal to a third value (e.g. 2), it is determined that the chroma deblocking offsets are not signaled and the value of the chroma deblocking offsets are inferred as in the current VVC draft standard. This may mean that the value in the SH is inferred from the PH and that the value of the PH is inferred from the PPS. Alternatively, the value in the SH is inferred from the PPS. When the proposed syntax element is not present, we propose that the value is inferred to be equal to 2.

[00210] The proposed method is based on embodiment 1 in the way that when Syntax_A is not present, and the determined value of Syntax_D is equal to a first value (e.g. 0), the value of Syntax_A is inferred to be the value of Syntax_B, if the determined value of Syntax_D is equal to a second value (e.g. 2), the value of Syntax_A is inferred to be the value of Syntax_C, wherein the Syntax_A represents one of the chroma deblocking offsets. Here we use Syntax_A represents `ph_cb_beta_offset_div2` as an example to show the syntax construct pseudo code:

```
if ( Syntax_D == 1 )
{
    // Syntax_A is present in the bitstream
```

```

    Syntax_A = decoded value of Syntax_A in the bitstream
}
else
{
    // Syntax_A is not present in the bitstream
    if ( Syntax_D == 0 )
    {
        Syntax_A = Syntax_B
    }
    else // Syntax_D is equal to 2
    {
        Syntax_A = Syntax_C
    }
}
}
Where,
Syntax_A = ph_cb_beta_offset_div2
Syntax_B = ph_beta_offset_div2
Syntax_C = pps_cb_beta_offset_div2
Syntax_D = ph_deblocking_chroma_offsets_present_idc

```

[00211] In another version of this embodiment, at least one of Syntax_B, Syntax_C and Syntax_D is a constant value.

[00212] In another version of this embodiment, at least one of Syntax_B, Syntax_C and Syntax_D is a value derived from one or more syntax elements in the bitstream.

[00213] Below are example syntax and semantics. Here we use the PH as an example.

TABLE 19

pic_parameter_set_rbsp() {	Descriptor
...	...
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
pps_deblocking_filter_disabled_flag OR pps_deblocking_filter_enabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag OR	
pps_deblocking_filter_enabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)

pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
...	...
rbsp_trailing_bits()	
}	

[00214] deblocking_filter_control_present_flag equal to 1 specifies the presence of deblocking filter control syntax elements in the PPS. **deblocking_filter_control_present_flag** equal to 0 specifies the absence of deblocking filter control syntax elements in the PPS.

[00215] deblocking_filter_override_enabled_flag equal to 1 specifies the presence of **ph_deblocking_filter_override_flag** in the PHs referring to the PPS or **slice_deblocking_filter_override_flag** in the slice headers referring to the PPS. **deblocking_filter_override_enabled_flag** equal to 0 specifies the absence of **ph_deblocking_filter_override_flag** in PHs referring to the PPS or **slice_deblocking_filter_override_flag** in slice headers referring to the PPS. When not present, the value of **deblocking_filter_override_enabled_flag** is inferred to be equal to 0.

[00216] pps_deblocking_filter_disabled_flag equal to 1 specifies that the operation of deblocking filter is not applied for slices referring to the PPS in which **slice_deblocking_filter_disabled_flag** is not present. **pps_deblocking_filter_disabled_flag** equal to 0 specifies that the operation of the deblocking filter is applied for slices referring to the PPS in which **slice_deblocking_filter_disabled_flag** is not present. When not present, the value of **pps_deblocking_filter_disabled_flag** is inferred to be equal to 0.

[00217] pps_beta_offset_div2 and **pps_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of **pps_beta_offset_div2** and **pps_tc_offset_div2** shall both be in the range of -12 to 12 , inclusive. When not present, the values of **pps_beta_offset_div2** and **pps_tc_offset_div2** are both inferred to be equal to 0.

[00218] **pps_cb_beta_offset_div2** and **pps_cb_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of **pps_cb_beta_offset_div2** and **pps_cb_tc_offset_div2** shall both be in the range of -12 to 12 , inclusive. When not present, the values of **pps_cb_beta_offset_div2** and **pps_cb_tc_offset_div2** are both inferred to be equal to 0.

[00219] **pps_cr_beta_offset_div2** and **pps_cr_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of **pps_cr_beta_offset_div2** and **pps_cr_tc_offset_div2** shall both be in the range of -12 to 12 , inclusive. When not present, the values of **pps_cr_beta_offset_div2** and **pps_cr_tc_offset_div2** are both inferred to be equal to 0.

TABLE 20

picture_header_structure() {	Descriptor
...	...
if(deblocking_filter_override_enabled_flag && dbf_info_in_ph_flag) {	
ph_deblocking_filter_override_flag	u(1)
if(ph_deblocking_filter_override_flag) {	
ph_deblocking_filter_disabled_flag OR ph_deblocking_filter_enabled_flag	u(1)
if(!ph_deblocking_filter_disabled_flag OR ph_deblocking_filter_enabled_flag)	
{	
ph_beta_offset_div2	se(v)
ph_tc_offset_div2	se(v)
if(ChromaArrayType != 0) {	
ph_deblocking_chroma_offsets_present_idc	u(1)
if(ph_deblocking_chroma_offsets_present_idc == 1) {	
ph_cb_beta_offset_div2	se(v)
ph_cb_tc_offset_div2	se(v)
ph_cr_beta_offset_div2	se(v)
ph_cr_tc_offset_div2	se(v)
}	
}	
}	

}	
}	
}	
}	

[00220] **ph_deblocking_filter_override_flag** equal to 1 specifies that deblocking parameters are present in the PH. **ph_deblocking_filter_override_flag** equal to 0 specifies that deblocking parameters are not present in the PH. When not present, the value of **ph_deblocking_filter_override_flag** is inferred to be equal to 0.

[00221] **ph_deblocking_filter_disabled_flag** equal to 1 specifies that the operation of the deblocking filter is not applied for the slices associated with the PH. **ph_deblocking_filter_disabled_flag** equal to 0 specifies that the operation of the deblocking filter is applied for the slices associated with the PH. When **ph_deblocking_filter_disabled_flag** is not present, it is inferred to be equal to **pps_deblocking_filter_disabled_flag**. As noted above, in some embodiments, the **ph_deblocking_filter_disabled_flag** is replaced with the **ph_deblocking_filter_enabled_flag**.

[00222] **ph_beta_offset_div2** and **ph_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for the slices associated with the PH. The values of **ph_beta_offset_div2** and **ph_tc_offset_div2** shall both be in the range of -12 to 12 , inclusive. When not present, the values of **ph_beta_offset_div2** and **ph_tc_offset_div2** are inferred to be equal to **pps_beta_offset_div2** and **pps_tc_offset_div2**, respectively.

[00223] **ph_deblocking_chroma_offsets_present_idc** equal to 1 specifies the presence of deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb and Cr components in the PH. **ph_deblocking_chroma_offsets_present_idc** equal to 0 or 2 specifies the absence of deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb and Cr components in the PH. When not present, the value of **ph_deblocking_chroma_offsets_present_idc** is inferred to be equal to 2.

[00224] **ph_cb_beta_offset_div2** and **ph_cb_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for the slices associated with the PH. The values of **ph_cb_beta_offset_div2** and **ph_cb_tc_offset_div2** shall

both be in the range of -12 to 12 , inclusive. When not present, the values of `ph_cb_beta_offset_div2` and `ph_cb_tc_offset_div2` are inferred as follows: i) When `ph_deblocking_chroma_offsets_present_idc` is equal to 0 , the values of `ph_cb_beta_offset_div2` and `ph_cb_tc_offset_div2` are inferred to be equal to `ph_beta_offset_div2` and `ph_tc_offset_div2`, respectively; and 2) When `ph_deblocking_chroma_offsets_present_idc` is equal to 2 , the values of `ph_cb_beta_offset_div2` and `ph_cb_tc_offset_div2` are inferred to be equal to `pps_cb_beta_offset_div2` and `pps_cb_tc_offset_div2`, respectively.

[00225] `ph_cr_beta_offset_div2` and `ph_cr_tc_offset_div2` specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for the slices associated with the PH. The values of `ph_cr_beta_offset_div2` and `ph_cr_tc_offset_div2` shall both be in the range of -12 to 12 , inclusive. When not present, the values of `ph_cr_beta_offset_div2` and `ph_cr_tc_offset_div2` are inferred as follows: 1) When `ph_deblocking_chroma_offsets_present_idc` is equal to 0 , the values of `ph_cr_beta_offset_div2` and `ph_cr_tc_offset_div2` are inferred to be equal to `ph_beta_offset_div2` and `ph_tc_offset_div2`, respectively; and 2) When `ph_deblocking_chroma_offsets_present_idc` is equal to 2 , the values of `ph_cr_beta_offset_div2` and `ph_cr_tc_offset_div2` are inferred to be equal to `pps_cr_beta_offset_div2` and `pps_cr_tc_offset_div2`, respectively.

[00226] Embodiment 2 – An example of disable the operation of the deblocking filter to the chroma components while the operation of the deblocking filter to the luma component is enabled

[00227] It is proposed to add syntax element in the PPS, PH and SH to disable the operation of the deblocking filter to the chroma components while the operation of the deblocking filter to the luma component is enabled. Embodiment 2 may be combined with Embodiment 1.

[00228] Below are example syntax and semantics in embodiment 2.

TABLE 21

pic_parameter_set_rbsp() {	Descriptor
...	...
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)

pps_deblocking_filter_disabled_flag OR pps_deblocking_filter_enabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag OR pps_deblocking_filter_enabled_flag) {	
pps_chroma_deblocking_filter_disabled_flag OR OR pps_chroma_deblocking_filter_enabled_flag	u(1)
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
if(!pps_chroma_deblocking_filter_disabled_flag OR pps_chroma_deblocking_filter_enabled_flag) {	
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	
}	
}	
rbsp_trailing_bits()	
}	

[00229] deblocking_filter_control_present_flag equal to 1 specifies the presence of deblocking filter control syntax elements in the PPS. **deblocking_filter_control_present_flag** equal to 0 specifies the absence of deblocking filter control syntax elements in the PPS.

[00230] deblocking_filter_override_enabled_flag equal to 1 specifies the presence of **ph_deblocking_filter_override_flag** in the PHs referring to the PPS or **slice_deblocking_filter_override_flag** in the slice headers referring to the PPS. **deblocking_filter_override_enabled_flag** equal to 0 specifies the absence of **ph_deblocking_filter_override_flag** in PHs referring to the PPS or **slice_deblocking_filter_override_flag** in slice headers referring to the PPS. When not present, the value of **deblocking_filter_override_enabled_flag** is inferred to be equal to 0.

[00231] pps_deblocking_filter_disabled_flag equal to 1 specifies that the operation of deblocking filter is not applied for slices referring to the PPS in which **slice_deblocking_filter_disabled_flag** is not present. **pps_deblocking_filter_disabled_flag** equal to 0 specifies that the operation of the deblocking filter is applied for slices referring to the PPS in which **slice_deblocking_filter_disabled_flag** is not present. When not present, the value of

pps_deblocking_filter_disabled_flag is inferred to be equal to 0. As noted above, the pps_deblocking_filter_disabled_flag may be replaced with the pps_deblocking_filter_enabled_flag.

[00232] pps_chroma_deblocking_filter_disabled_flag equal to 1 specifies that the operation of deblocking filter to the chroma components is not applied for slices referring to the PPS in which slice_chroma_deblocking_filter_disabled_flag is not present.

pps_chroma_deblocking_filter_disabled_flag equal to 0 specifies that the operation of the deblocking filter to the chroma components is applied for slices referring to the PPS in which slice_chroma_deblocking_filter_disabled_flag is not present. When not present, the value of pps_chroma_deblocking_filter_disabled_flag is inferred to be equal to 0. The pps_chroma_deblocking_filter_disabled_flag may be replaced with the pps_chroma_deblocking_filter_enabled_flag.

[00233] pps_beta_offset_div2 and pps_tc_offset_div2 specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps_beta_offset_div2 and pps_tc_offset_div2 shall both be in the range of -12 to 12 , inclusive. When not present, the values of pps_beta_offset_div2 and pps_tc_offset_div2 are both inferred to be equal to 0.

[00234] pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 shall both be in the range of -12 to 12 , inclusive. When not present, the values of pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 are both inferred to be equal to pps_beta_offset_div2 and pps_tc_offset_div2, respectively. $\oplus$.

[00235] pps_cr_beta_offset_div2 and pps_cr_tc_offset_div2 specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for slices

referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the picture headers or the slice headers of the slices referring to the PPS. The values of `pps_cr_beta_offset_div2` and `pps_cr_tc_offset_div2` shall both be in the range of -12 to 12 , inclusive. When not present, the values of `pps_cr_beta_offset_div2` and `pps_cr_tc_offset_div2` are both inferred to be equal to `pps_beta_offset_div2` and `pps_tc_offset_div2`, respectively.

TABLE 22

picture_header_structure() {	Descriptor
...	...
if(deblocking_filter_override_enabled_flag && dbf_info_in_ph_flag) {	
ph_deblocking_filter_override_flag	u(1)
if(ph_deblocking_filter_override_flag) {	
ph_deblocking_filter_disabled_flag OR <code>ph_deblocking_filter_enabled_flag</code>	u(1)
if(!ph_deblocking_filter_disabled_flag OR <code>ph_deblocking_filter_enabled_flag</code>)	
{	
ph_chroma_deblocking_filter_disabled_flag OR ph_chroma_deblocking_filter_enabled_flag	u(1)
ph_beta_offset_div2	se(v)
ph_tc_offset_div2	se(v)
if(!ph_chroma_deblocking_filter_disabled_flag OR ph_chroma_deblocking_filter_enabled_flag) {	
ph_cb_beta_offset_div2	se(v)
ph_cb_tc_offset_div2	se(v)
ph_cr_beta_offset_div2	se(v)
ph_cr_tc_offset_div2	se(v)
}	
}	
}	
}	
...	...

[00236] **ph_deblocking_filter_override_flag** equal to 1 specifies that deblocking parameters are present in the PH. `ph_deblocking_filter_override_flag` equal to 0 specifies that deblocking parameters are not present in the PH. When not present, the value of `ph_deblocking_filter_override_flag` is inferred to be equal to 0.

[00237] **ph_deblocking_filter_disabled_flag** equal to 1 specifies that the operation of the deblocking filter is not applied for the slices associated with the PH.

ph_deblocking_filter_disabled_flag equal to 0 specifies that the operation of the deblocking filter is applied for the slices associated with the PH. When ph_deblocking_filter_disabled_flag is not present, it is inferred to be equal to pps_deblocking_filter_disabled_flag. As noted above, in some embodiments, the ph_deblocking_filter_disabled_flag is replaced with the ph_deblocking_filter_enabled_flag.

[00238] **ph_chroma_deblocking_filter_disabled_flag** equal to 1 specifies that the operation of the deblocking filter to the chroma components is not applied for the slices associated with the PH. ph_chroma_deblocking_filter_disabled_flag equal to 0 specifies that the operation of the deblocking filter to the chroma components is applied for the slices associated with the PH. When ph_chroma_deblocking_filter_disabled_flag is not present, it is inferred to be equal to pps_chroma_deblocking_filter_disabled_flag. In some embodiments, the ph_chroma_deblocking_filter_disabled_flag is replaced with the ph_chroma_deblocking_filter_enabled_flag.

[00239] **ph_beta_offset_div2** and **ph_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for the slices associated with the PH. The values of ph_beta_offset_div2 and ph_tc_offset_div2 shall both be in the range of -12 to 12 , inclusive. When not present, the values of ph_beta_offset_div2 and ph_tc_offset_div2 are inferred to be equal to pps_beta_offset_div2 and pps_tc_offset_div2, respectively.

[00240] **ph_cb_beta_offset_div2** and **ph_cb_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for the slices associated with the PH. The values of ph_cb_beta_offset_div2 and ph_cb_tc_offset_div2 shall both be in the range of -12 to 12 , inclusive. When not present, the values of ph_cb_beta_offset_div2 and ph_cb_tc_offset_div2 are inferred to be equal to pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2, respectively.

[00241] **ph_cr_beta_offset_div2** and **ph_cr_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for the slices associated with the PH. The values of ph_cr_beta_offset_div2 and ph_cr_tc_offset_div2 shall both be in the range of -12 to 12 , inclusive. When not present, the values of

ph_cr_beta_offset_div2 and ph_cr_tc_offset_div2 are inferred to be equal to pps_cr_beta_offset_div2 and pps_cr_tc_offset_div2, respectively.

TABLE 23

slice_header() {	Descriptor
...	...
if(deblocking_filter_override_enabled_flag && !dbf_info_in_ph_flag)	
slice_deblocking_filter_override_flag	u(1)
if(slice_deblocking_filter_override_flag) {	
slice_deblocking_filter_disabled_flag OR slice_deblocking_filter_enabled_flag	u(1)
if(!slice_deblocking_filter_disabled_flag OR	
slice_deblocking_filter_enabled_flag) {	
slice_chroma_deblocking_filter_disabled_flag OR slice_chroma_deblocking_filter_enabled_flag	u(1)
slice_beta_offset_div2	se(v)
slice_tc_offset_div2	se(v)
if(!slice_chroma_deblocking_filter_disabled_flag OR	
slice_chroma_deblocking_filter_enabled_flag) {	
slice_cb_beta_offset_div2	se(v)
slice_cb_tc_offset_div2	se(v)
slice_cr_beta_offset_div2	se(v)
slice_cr_tc_offset_div2	se(v)
}	
}	
}	
...	...
byte_alignment()	
}	

[00242] **slice_deblocking_filter_override_flag** equal to 1 specifies that deblocking parameters are present in the slice header. **slice_deblocking_filter_override_flag** equal to 0 specifies that deblocking parameters are not present in the slice header. When not present, the value of **slice_deblocking_filter_override_flag** is inferred to be equal to **ph_deblocking_filter_override_flag**.

[00243] **slice_deblocking_filter_disabled_flag** equal to 1 specifies that the operation of the deblocking filter is not applied for the current slice. **slice_deblocking_filter_disabled_flag** equal to 0 specifies that the operation of the deblocking filter is applied for the current slice. When

slice_deblocking_filter_disabled_flag is not present, it is inferred to be equal to ph_deblocking_filter_disabled_flag or !ph_deblocking_filter_enabled_flag. In some embodiments, the slice_deblocking_filter_disabled_flag is replaced with the slice_deblocking_filter_enabled_flag.

[00244] slice_chroma_deblocking_filter_disabled_flag equal to 1 specifies that the operation of the deblocking filter to the chroma components is not applied for the current slice. slice_chroma_deblocking_filter_disabled_flag equal to 0 specifies that the operation of the deblocking filter to the chroma components is applied for the current slice. When slice_chroma_deblocking_filter_disabled_flag is not present, it is inferred to be equal to ph_chroma_deblocking_filter_disabled_flag. In some embodiments, the slice_chroma_deblocking_filter_disabled_flag is replaced with the slice_chroma_deblocking_filter_enabled_flag.

[00245] slice_beta_offset_div2 and slice_tc_offset_div2 specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the luma component for the current slice. The values of slice_beta_offset_div2 and slice_tc_offset_div2 shall both be in the range of -12 to 12, inclusive. When not present, the values of slice_beta_offset_div2 and slice_tc_offset_div2 are inferred to be equal to ph_beta_offset_div2 and ph_tc_offset_div2, respectively.

[00246] slice_cb_beta_offset_div2 and slice_cb_tc_offset_div2 specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cb component for the current slice. The values of slice_cb_beta_offset_div2 and slice_cb_tc_offset_div2 shall both be in the range of -12 to 12, inclusive. When not present, the values of slice_cb_beta_offset_div2 and slice_cb_tc_offset_div2 are inferred to be equal to ph_cb_beta_offset_div2 and ph_cb_tc_offset_div2, respectively.

[00247] slice_cr_beta_offset_div2 and slice_cr_tc_offset_div2 specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to the Cr component for the current slice. The values of slice_cr_beta_offset_div2 and slice_cr_tc_offset_div2 shall both be in the range of -12 to 12, inclusive. When not present, the values of slice_cr_beta_offset_div2 and slice_cr_tc_offset_div2 are inferred to be equal to ph_cr_beta_offset_div2 and ph_cr_tc_offset_div2, respectively.

[00248] Embodiment 3

[00249] As illustrated above, the presence of syntax elements that are applied to two chroma components may be controlled by a one-bit syntax element. In another words, in some embodiments, the presence of Cr follows the presence of Cb.

[00250] In this embodiment a first one-bit syntax element is used for the Cb component and a second one-bit syntax element is used for the Cr component separately to control the present of the syntax elements that are applied to the Cb component and to the Cr component.

[00251] In a variant of this embodiment a two-bits syntax element is used to jointly control the present of the syntax element that are applied to the Cb component and to the Cr component, as described below.

[00252] If the value of two-bit syntax element is equal to 0 it specifies the absence of the syntax elements that are applied to the Cb and Cr components.

[00253] If the value of two-bit syntax element is equal to 1 it specifies the presence of the syntax elements that are applied to the Cb and the absence of the syntax elements that are applied to the Cr components.

[00254] If the value of two-bit syntax element is equal to 2 it specifies the absence of the syntax elements that are applied to the Cb and the presence of the syntax elements that are applied to the Cr components.

[00255] If the value of two-bit syntax element is equal to 3 it specifies the presence of the syntax elements that are applied to the Cb and Cr components.

[00256] Embodiment 4 – Differentiating based on inference to X and signaling as X

[00257] In this embodiment a condition method is proposed that differentiates between the inference of the value of the syntax element A to the value X and explicit signaling of syntax element A equal to the value X. In one example of the current embodiment, the control of the Cb component and Cr component may be defined as follows.

[00258] If the value of syntax element A is signaled as a first value, it specifies the absence of the syntax elements that are applied to the Cb and Cr components.

[00259] If the value of syntax element A is inferred as a first value, it specifies the presence of the syntax elements that are applied to the Cb and the absence of the syntax elements that are applied to the Cr components.

[00260] If the value of syntax element A is signaled as a second value, it specifies the absence of the syntax elements that are applied to the Cb and the presence of the syntax elements that are applied to the Cr components.

[00261] If the value of syntax element A is inferred as a second value, it specifies the presence of the syntax elements that are applied to the Cb and Cr components.

[00262] FIG. 12 is a block diagram of an apparatus 1200 for implementing decoder 204 and/or encoder 202, according to some embodiments. When apparatus 1200 implements a decoder, apparatus 1200 may be referred to as a “decoding apparatus 1200,” and when apparatus 1200 implements an encoder, apparatus 1200 may be referred to as an “encoding apparatus 1200.” As shown in FIG. 12, apparatus 1200 may comprise: processing circuitry (PC) 1202, which may include one or more processors (P) 1255 (e.g., a general purpose microprocessor and/or one or more other processors, such as an application specific integrated circuit (ASIC), field-programmable gate arrays (FPGAs), and the like), which processors may be co-located in a single housing or in a single data center or may be geographically distributed (i.e., apparatus 1200 may be a distributed computing apparatus); at least one network interface 1248 comprising a transmitter (Tx) 1245 and a receiver (Rx) 1247 for enabling apparatus 1200 to transmit data to and receive data from other nodes connected to a network 110 (e.g., an Internet Protocol (IP) network) to which network interface 1248 is connected (directly or indirectly) (e.g., network interface 1248 may be wirelessly connected to the network 110, in which case network interface 1248 is connected to an antenna arrangement); and a storage unit (a.k.a., “data storage system”) 1208, which may include one or more non-volatile storage devices and/or one or more volatile storage devices. In embodiments where PC 1202 includes a programmable processor, a computer program product (CPP) 1241 may be provided. CPP 1241 includes a computer readable medium (CRM) 1242 storing a computer program (CP) 1243 comprising computer readable instructions (CRI) 1244. CRM 1242 may be a non-transitory computer readable medium, such as, magnetic media (e.g., a hard disk), optical media, memory devices (e.g., random access memory, flash memory), and the like. In some embodiments, the CRI 1244 of

computer program 1243 is configured such that when executed by PC 1202, the CRI causes apparatus 1200 to perform steps described herein (e.g., steps described herein with reference to the flow charts). In other embodiments, apparatus 1200 may be configured to perform steps described herein without the need for code. That is, for example, PC 1202 may consist merely of one or more ASICs. Hence, the features of the embodiments described herein may be implemented in hardware and/or software.

[00263] While various embodiments are described herein, it should be understood that they have been presented by way of example only, and not limitation. Thus, the breadth and scope of this disclosure should not be limited by any of the above-described exemplary embodiments. Moreover, any combination of the above-described elements in all possible variations thereof is encompassed by the disclosure unless otherwise indicated herein or otherwise clearly contradicted by context.

[00264] Additionally, while the processes described above and illustrated in the drawings are shown as a sequence of steps, this was done solely for the sake of illustration. Accordingly, it is contemplated that some steps may be added, some steps may be omitted, the order of the steps may be re-arranged, and some steps may be performed in parallel.

CLAIMS

1. A method, performed by a decoder, for deriving a value for a first syntax element, Syntax_A, the method comprising:
 - determining whether Syntax_A is present in a bitstream;
 - deriving a value D from at least a value for a fourth syntax element, Syntax_D;
 - as a result of determining that Syntax_A is not present in the bitstream:
 - deriving a value B from at least a value for a second syntax element, Syntax_B;
 - deriving the value for Syntax_A to be equal to the value B, if D is equal to X,
 - or:
 - deriving a value C from at least a value for a third syntax element, Syntax_C;
 - deriving the value for Syntax_A to be equal to the value C, if D is equal to Y,
 - where Y does not equal X, and
 - decoding a picture from the bitstream using the derived value for Syntax_A.
2. The method of claim 1, where at least one of B, C and D is equal to the value for the syntax element Syntax_B, Syntax_C, and Syntax_D, respectively.
3. The method of any of claims 1-2, wherein at least one of Syntax_B, Syntax_C and Syntax_D is decoded from the bitstream.
4. The method of any of claims 1-3, wherein deriving the value B comprises:
 - determining whether the second syntax element, Syntax_B, is present in the bitstream;
 - and
 - as a result of determining that Syntax_B is not present in the bitstream, inferring the value B, otherwise decoding the value B from Syntax_B.
5. The method of any of claims 1-4, wherein deriving the value D comprises:
 - determining whether the fourth syntax element, Syntax_D, is present in the bitstream;
 - and

if Syntax_D is determined present in the bitstream, setting D equal to a third value, wherein the third value is a predetermined value or the third value is equal to the decoded value of Syntax_D.

6. The method of claim 5, wherein deriving the value D comprises setting D equal to a fourth value if Syntax_D is determined to be not present in the bitstream, where the fourth value is a predetermined value or the fourth value is equal to an inferred value of Syntax_D.

7. The method of any of claims 1-6, wherein at least one of the Syntax_A, Syntax_B, Syntax_C or Syntax_D is a deblocking parameter.

8. The method of any of claims 1-7, wherein all said existing syntax elements, Syntax_A, Syntax_B, Syntax_C and Syntax_D, are in one header or parameter set.

9. The method of any of claims 1-7, wherein all said existing syntax elements, Syntax_A, Syntax_B, Syntax_C and Syntax_D, are in several headers or parameter sets.

10. A computer program (1243) comprising instructions (1244) which when executed by processing circuitry (1202) causes the processing circuitry (1202) to perform the method of any of claims 1-9.

11. A carrier comprising the computer program of claim 10, wherein the carrier is one of an electronic signal, an optical signal, a radio signal, and a computer readable storage medium (1242).

12. An apparatus (1200), the apparatus being adapted to perform the method of any one of claims 1-9.

13. An apparatus (1200), the apparatus comprising:
processing circuitry (1202); and

a memory (1242), said memory containing instructions (1244) executable by said processing circuitry, whereby said apparatus is operative to perform the method of any of claims 1-9.