

SIGNALING OF GENERAL CONSTRAIN INFORMATION

CROSS-REFERENCE TO RELATED APPLICATION

[001] Under the applicable patent law and/or rules pursuant to the Paris Convention, this application is made to timely claim the priority to and benefits of U.S. Provisional Patent Application No. 63/033,689 filed on June 2, 2020. For all purposes under the law, the entire disclosure of the aforementioned application is incorporated by reference as part of the disclosure of this application.

TECHNICAL FIELD

[002] This patent document relates to image and video coding and decoding.

BACKGROUND

[003] Digital video accounts for the largest bandwidth use on the internet and other digital communication networks. As the number of connected user devices capable of receiving and displaying video increases, it is expected that the bandwidth demand for digital video usage will continue to grow.

SUMMARY

[004] The present document discloses techniques for the signaling of general constraint information (GCI) that can be used by video encoders and decoders to perform video encoding, decoding, or processing.

[005] In one example aspect, a video processing method is disclosed. The method includes performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies that a syntax structure in a profile-tier-level syntax structure is after a syntax element, wherein the syntax structure comprises information related to general constraint information (GCI) for the bitstream, and wherein the syntax element indicates a level to which an output layer set associated with the profile-tier-level syntax structure conforms.

[006] In another example aspect, another video processing method is disclosed. The method includes performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies that a byte alignment syntax

in a general constraint information (GCI) syntax structure is after one or more GCI reserved fields, wherein the byte alignment syntax indicates whether a current position in the bitstream is an integer multiple of 8 bits from a position of a first bit in the bitstream, and wherein the GCI syntax structure comprises GCI related syntax elements.

[007] In yet another example aspect, another video processing method is disclosed. The method includes performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies that a syntax structure in a profile-tier-level syntax structure is after an indication of level information, wherein the syntax structure comprises information related to general constraint information (GCI), and wherein the indication of level information specifies an interoperability indicator.

[008] In yet another example aspect, another video processing method is disclosed. The method includes performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein rule specifies that a syntax element in a profile-tier-level syntax structure indicates whether a general constraint information (GCI) syntax structure is included in the profile-tier-level syntax structure.

[009] In yet another example aspect, another video processing method is disclosed. The method includes performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies that a byte alignment syntax is excluded from a general constraint information (GCI) syntax structure that is present in the profile-tier-level syntax structure, wherein the byte alignment syntax indicates whether a current position in the bitstream is an integer multiple of 8 bits from a position of a first bit in the bitstream, and wherein the GCI syntax structure comprises a GCI related syntax element.

[0010] In yet another example aspect, another video processing method is disclosed. The method includes performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies that a general constraint information (GCI) syntax structure is immediately before a byte alignment checking condition in a profile-tier-level syntax structure, wherein the GCI syntax structure comprises GCI related syntax elements, and wherein the byte alignment checking condition checks whether a current position in the bitstream is an integer multiple of 8 bits from a position of a first bit in the bitstream.

[0011] In yet another example aspect, another video processing method is disclosed. The method includes performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies that a number of a plurality of reserved constraint bits associated with a general constraint information (GCI) syntax element is included in the bitstream.

[0012] In yet another example aspect, another video processing method is disclosed. The method includes performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies that a general constraint information (GCI) syntax element is included at a beginning of a GCI syntax structure, wherein the GCI syntax element indicates whether one or more GCI syntax elements are included in the GCI syntax structure.

[0013] In yet another example aspect, another video processing method is disclosed. The method includes performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies a constraint on a syntax element, wherein the syntax element corresponds to a bit depth used for representing the video in the bitstream.

[0014] In yet another example aspect, another video processing method is disclosed. The method includes performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies a constraint on a syntax element, wherein the syntax element corresponds to a chroma format of the video.

[0015] In yet another example aspect, a video encoder apparatus is disclosed. The video encoder comprises a processor configured to implement above-described methods.

[0016] In yet another example aspect, a video decoder apparatus is disclosed. The video decoder comprises a processor configured to implement above-described methods.

[0017] In yet another example aspect, a computer readable medium having code stored thereon is disclosed. The code embodies one of the methods described herein in the form of processor-executable code.

[0018] These, and other, features are described throughout the present document.

BRIEF DESCRIPTION OF DRAWINGS

[0019] FIG. 1 is a block diagram showing an example video processing system in which various techniques disclosed herein may be implemented.

[0020] FIG. 2 is a block diagram of an example hardware platform used for video processing.

[0021] FIG. 3 is a block diagram that illustrates an example video coding system that can implement some embodiments of the present disclosure.

[0022] FIG. 4 is a block diagram that illustrates an example of an encoder that can implement some embodiments of the present disclosure.

[0023] FIG. 5 is a block diagram that illustrates an example of a decoder that can implement some embodiments of the present disclosure.

[0024] FIGS. 6-15 show flowcharts for example methods of video processing.

DETAILED DESCRIPTION

[0025] Section headings are used in the present document for ease of understanding and do not limit the applicability of techniques and embodiments disclosed in each section only to that section. Furthermore, H.266 terminology is used in some description only for ease of understanding and not for limiting scope of the disclosed techniques. As such, the techniques described herein are applicable to other video codec protocols and designs also.

1. Introduction

[0026] This document is related to video coding technologies. Specifically, it is about syntax designs for signaling of general constraints information (GCI) in video coding. The ideas may be applied individually or in various combination, to any video coding standard or non-standard video codec that supports multi-layer video coding, e.g., the being-developed Versatile Video Coding (VVC).

2. Abbreviations

APS	Adaptation Parameter Set
AU	Access Unit
AUD	Access Unit Delimiter
AVC	Advanced Video Coding
CLVS	Coded Layer Video Sequence
CPB	Coded Picture Buffer

CRA	Clean Random Access
CTU	Coding Tree Unit
CVS	Coded Video Sequence
DPB	Decoded Picture Buffer
DPS	Decoding Parameter Set
EOB	End Of Bitstream
EOS	End Of Sequence
GCI	General Constraints Information
GDR	Gradual Decoding Refresh
HEVC	High Efficiency Video Coding
HRD	Hypothetical Reference Decoder
IDR	Instantaneous Decoding Refresh
JEM	Joint Exploration Model
MCTS	Motion-Constrained Tile Sets
NAL	Network Abstraction Layer
OLS	Output Layer Set
PH	Picture Header
PPS	Picture Parameter Set
PTL	Profile, Tier and Level
PU	Picture Unit
RRP	Reference Picture Resampling
RBSP	Raw Byte Sequence Payload
SEI	Supplemental Enhancement Information
SH	Slice Header
SPS	Sequence Parameter Set
SVC	Scalable Video Coding
VCL	Video Coding Layer
VPS	Video Parameter Set
VTM	VVC Test Model
VUI	Video Usability Information
VVC	Versatile Video Coding

3. Initial discussion

[0027] Video coding standards have evolved primarily through the development of the well-known ITU-T and ISO/IEC standards. The ITU-T produced H.261 and H.263, ISO/IEC produced MPEG-1 and MPEG-4 Visual, and the two organizations jointly produced the H.262/MPEG-2 Video and H.264/MPEG-4 Advanced Video Coding (AVC) and H.265/HEVC standards. Since H.262, the video coding standards are based on the hybrid video coding structure wherein temporal prediction plus transform coding are utilized. To explore the future video coding technologies beyond HEVC, the Joint Video Exploration Team (JVET) was founded by VCEG and MPEG jointly in 2015. Since then, many new methods have been adopted by JVET and put into the reference software named Joint Exploration Model (JEM). The JVET meeting is concurrently held once every quarter, and the new coding standard is targeting at 50% bitrate reduction as compared to HEVC. The new video coding standard was officially named as Versatile Video Coding (VVC) in the April 2018 JVET meeting, and the first version of VVC test model (VTM) was released at that time. As there are continuous effort contributing to VVC standardization, new coding techniques are being adopted to the VVC standard in every JVET meeting. The VVC working draft and test model VTM are then updated after every meeting. The VVC project is now aiming for technical completion (FDIS) at the July 2020 meeting.

3.1. Picture resolution change within a sequence

[0028] In AVC and HEVC, the spatial resolution of pictures cannot change unless a new sequence using a new SPS starts, with an IRAP picture. VVC enables picture resolution change within a sequence at a position without encoding an IRAP picture, which is always intra-coded. This feature is sometimes referred to as reference picture resampling (RPR), as the feature needs resampling of a reference picture used for inter prediction when that reference picture has a different resolution than the current picture being decoded.

[0029] The scaling ratio is restricted to be larger than or equal to $1/2$ (2 times downsampling from the reference picture to the current picture), and less than or equal to 8 (8 times upsampling). Three sets of resampling filters with different frequency cutoffs are specified to handle various scaling ratios between a reference picture and the current picture. The three sets of resampling filters are applied respectively for the scaling ratio ranging from $1/2$ to $1/1.75$, from $1/1.75$ to $1/1.25$, and from $1/1.25$ to 8. Each set of resampling filters has 16 phases for luma

and 32 phases for chroma which is same to the case of motion compensation interpolation filters. Actually the normal MC interpolation process is a special case of the resampling process with scaling ratio ranging from 1/1.25 to 8. The horizontal and vertical scaling ratios are derived based on picture width and height, and the left, right, top and bottom scaling offsets specified for the reference picture and the current picture.

[0030] Other aspects of the VVC design for support of this feature that are different from HEVC include: i) The picture resolution and the corresponding conformance window are signaled in the PPS instead of in the SPS, while in the SPS the maximum picture resolution is signaled. ii) For a single-layer bitstream, each picture store (a slot in the DPB for storage of one decoded picture) occupies the buffer size as required for storing a decoded picture having the maximum picture resolution.

3.2. Scalable video coding (SVC) in general and in VVC

[0031] Scalable video coding (SVC, sometimes also just referred to as scalability in video coding) refers to video coding in which a base layer (BL), sometimes referred to as a reference layer (RL), and one or more scalable enhancement layers (ELs) are used. In SVC, the base layer can carry video data with a base level of quality. The one or more enhancement layers can carry additional video data to support, for example, higher spatial, temporal, and/or signal-to-noise (SNR) levels. Enhancement layers may be defined relative to a previously encoded layer. For example, a bottom layer may serve as a BL, while a top layer may serve as an EL. Middle layers may serve as either ELs or RLs, or both. For example, a middle layer (e.g., a layer that is neither the lowest layer nor the highest layer) may be an EL for the layers below the middle layer, such as the base layer or any intervening enhancement layers, and at the same time serve as a RL for one or more enhancement layers above the middle layer. Similarly, in the Multiview or 3D extension of the HEVC standard, there may be multiple views, and information of one view may be utilized to code (e.g., encode or decode) the information of another view (e.g., motion estimation, motion vector prediction and/or other redundancies).

[0032] In SVC, the parameters used by the encoder or the decoder are grouped into parameter sets based on the coding level (e.g., video-level, sequence-level, picture-level, slice level, etc.) in which they may be utilized. For example, parameters that may be utilized by one or more coded video sequences of different layers in the bitstream may be included in a video parameter set (VPS), and parameters that are utilized by one or more pictures in a coded video sequence may

be included in a sequence parameter set (SPS). Similarly, parameters that are utilized by one or more slices in a picture may be included in a picture parameter set (PPS), and other parameters that are specific to a single slice may be included in a slice header. Similarly, the indication of which parameter set(s) a particular layer is using at a given time may be provided at various coding levels.

[0033] Thanks to the support of reference picture resampling (RPR) in VVC, support of a bitstream containing multiple layers, e.g., two layers with SD and HD resolutions in VVC can be designed without the need any additional signal-processing-level coding tool, as upsampling needed for spatial scalability support can just use the RPR upsampling filter. Nevertheless, high-level syntax changes (compared to not supporting scalability) are needed for scalability support. Scalability support is specified in VVC version 1. Different from the scalability supports in any earlier video coding standards, including in extensions of AVC and HEVC, the design of VVC scalability has been made friendly to single-layer decoder designs as much as possible. The decoding capability for multi-layer bitstreams are specified in a manner as if there were only a single layer in the bitstream. E.g., the decoding capability, such as DPB size, is specified in a manner that is independent of the number of layers in the bitstream to be decoded. Basically, a decoder designed for single-layer bitstreams does not need much change to be able to decode multi-layer bitstreams. Compared to the designs of multi-layer extensions of AVC and HEVC, the HLS aspects have been significantly simplified at the sacrifice of some flexibilities. For example, an IRAP AU is required to contain a picture for each of the layers present in the CVS.

3.3. Parameter sets

[0034] AVC, HEVC, and VVC specify parameter sets. The types of parameter sets include SPS, PPS, APS, and VPS. SPS and PPS are supported in all of AVC, HEVC, and VVC. VPS was introduced since HEVC and is included in both HEVC and VVC. APS was not included in AVC or HEVC but is included in the latest VVC draft text.

[0035] SPS was designed to carry sequence-level header information, and PPS was designed to carry infrequently changing picture-level header information. With SPS and PPS, infrequently changing information need not to be repeated for each sequence or picture, hence redundant signaling of this information can be avoided. Furthermore, the use of SPS and PPS enables out-of-band transmission of the important header information, thus not only avoiding the need for redundant transmissions but also improving error resilience.

[0036] VPS was introduced for carrying sequence-level header information that is common for all layers in multi-layer bitstreams.

[0037] APS was introduced for carrying such picture-level or slice-level information that needs quite some bits to code, can be shared by multiple pictures, and in a sequence there can be quite many different variations.

3.4. General profile, tier, and level syntax and semantics

[0038] In the latest VVC draft text, the general profile, tier, level syntax and semantics are as follows:

profile_tier_level(profileTierPresentFlag, maxNumSubLayersMinus1) {	Descriptor
if(profileTierPresentFlag) {	
general_profile_idc	u(7)
general_tier_flag	u(1)
general_constraint_info()	
}	
general_level_idc	u(8)
if(profileTierPresentFlag) {	
ptl_num_sub_profiles	u(8)
for(i = 0; i < ptl_num_sub_profiles; i++)	
general_sub_profile_idc[i]	u(32)
}	
for(i = 0; i < maxNumSubLayersMinus1; i++)	
ptl_sublayer_level_present_flag[i]	u(1)
while(!byte_aligned())	
ptl_alignment_zero_bit	f(1)
for(i = 0; i < maxNumSubLayersMinus1; i++)	
if(ptl_sublayer_level_present_flag[i])	
sublayer_level_idc[i]	u(8)
}	

A profile_tier_level() syntax structure provides level information and, optionally, profile, tier, sub-profile, and general constraints information.

When the profile_tier_level() syntax structure is included in a VPS, the OlsInScope is one or more OLSs specified by the VPS. When the profile_tier_level() syntax structure is included in an SPS, the

OlsInScope is the OLS that includes only the layer that is the lowest layer among the layers that refer to the SPS, and this lowest layer is an independent layer.

general_profile_idc indicates a profile to which OlsInScope conforms as specified in Annex A.

Bitstreams shall not contain values of **general_profile_idc** other than those specified in Annex A. Other values of **general_profile_idc** are reserved for future use by ITU-T | ISO/IEC.

general_tier_flag specifies the tier context for the interpretation of **general_level_idc** as specified in Annex A.

general_level_idc indicates a level to which OlsInScope conforms as specified in Annex A. Bitstreams shall not contain values of **general_level_idc** other than those specified in Annex A. Other values of **general_level_idc** are reserved for future use by ITU-T | ISO/IEC.

NOTE 1 – A greater value of **general_level_idc** indicates a higher level. The maximum level signalled in the DCI NAL unit for OlsInScope may be higher than but cannot be lower than the level signalled in the SPS for a CLVS contained within OlsInScope.

NOTE 2 – When OlsInScope conforms to multiple profiles, **general_profile_idc** should indicate the profile that provides the preferred decoded result or the preferred bitstream identification, as determined by the encoder (in a manner not specified in this Specification).

NOTE 3 – When the CVSs of OlsInScope conform to different profiles, multiple **profile_tier_level()** syntax structures may be included in the DCI NAL unit such that for each CVS of the OlsInScope there is at least one set of indicated profile, tier, and level for a decoder that is capable of decoding the CVS.

ptl_num_sub_profiles specifies the number of the **general_sub_profile_idc[i]** syntax elements.

general_sub_profile_idc[i] specifies the indicator of the i-th interoperability metadata registered as specified by Rec. ITU-T T.35, the contents of which are not specified in this Specification.

ptl_sublayer_level_present_flag[i] equal to 1 specifies that level information is present in the **profile_tier_level()** syntax structure for the sublayer representation with TemporalId equal to i.

ptl_sublayer_level_present_flag[i] equal to 0 specifies that level information is not present in the **profile_tier_level()** syntax structure for the sublayer representation with TemporalId equal to i.

ptl_alignment_zero_bits shall be equal to 0.

The semantics of the syntax element **sublayer_level_idc[i]** is, apart from the specification of the inference of not present values, the same as the syntax element **general_level_idc**, but apply to the sublayer representation with TemporalId equal to i.

When not present, the value of **sublayer_level_idc[i]** is inferred as follows:

- sublayer_level_idc[maxNumSubLayersMinus1] is inferred to be equal to general_level_idc of the same profile_tier_level() structure,
- For i from maxNumSubLayersMinus1 – 1 to 0 (in decreasing order of values of i), inclusive, sublayer_level_idc[i] is inferred to be equal to sublayer_level_idc[i + 1].

3.5. General constraint information syntax and semantics

general_constraint_info() {	Descriptor
general_non_packed_constraint_flag	u(1)
general_frame_only_constraint_flag	u(1)
general_non_projected_constraint_flag	u(1)
general_one_picture_only_constraint_flag	u(1)
intra_only_constraint_flag	u(1)
max_bitdepth_minus8_constraint_idc	u(4)
max_chroma_format_constraint_idc	u(2)
single_layer_constraint_flag	u(1)
all_layers_independent_constraint_flag	u(1)
no_ref_pic_resampling_constraint_flag	u(1)
no_res_change_in_clvs_constraint_flag	u(1)
one_tile_per_pic_constraint_flag	u(1)
pic_header_in_slice_header_constraint_flag	u(1)
one_slice_per_pic_constraint_flag	u(1)
one_subpic_per_pic_constraint_flag	u(1)
no_qtbt_dual_tree_intra_constraint_flag	u(1)
no_partition_constraints_override_constraint_flag	u(1)
no_sao_constraint_flag	u(1)
no_alf_constraint_flag	u(1)
no_ccalf_constraint_flag	u(1)
no_joint_cbr_constraint_flag	u(1)
no_mrl_constraint_flag	u(1)
no_isp_constraint_flag	u(1)
no_mip_constraint_flag	u(1)
no_ref_wraparound_constraint_flag	u(1)
no_temporal_mvp_constraint_flag	u(1)

no_sbtmvp_constraint_flag	u(1)
no_amvr_constraint_flag	u(1)
no_bdof_constraint_flag	u(1)
no_dmvr_constraint_flag	u(1)
no_cclm_constraint_flag	u(1)
no_mts_constraint_flag	u(1)
no_sbt_constraint_flag	u(1)
no_lfnst_constraint_flag	u(1)
no_affine_motion_constraint_flag	u(1)
no_mmvd_constraint_flag	u(1)
no_smvd_constraint_flag	u(1)
no_prof_constraint_flag	u(1)
no_bcw_constraint_flag	u(1)
no_ibc_constraint_flag	u(1)
no_ciip_constraint_flag	u(1)
no_gpm_constraint_flag	u(1)
no_ladf_constraint_flag	u(1)
no_transform_skip_constraint_flag	u(1)
no_bdpcm_constraint_flag	u(1)
no_weighted_prediction_constraint_flag	u(1)
no_palette_constraint_flag	u(1)
no_act_constraint_flag	u(1)
no_lmcs_constraint_flag	u(1)
no_cu_qp_delta_constraint_flag	u(1)
no_chroma_qp_offset_constraint_flag	u(1)
no_dep_quant_constraint_flag	u(1)
no_sign_data_hiding_constraint_flag	u(1)
no_mixed_nalu_types_in_pic_constraint_flag	u(1)
no_trail_constraint_flag	u(1)
no_stsa_constraint_flag	u(1)
no_rasl_constraint_flag	u(1)
no_radl_constraint_flag	u(1)

no_idr_constraint_flag	u(1)
no_cra_constraint_flag	u(1)
no_gdr_constraint_flag	u(1)
no_aps_constraint_flag	u(1)
while(!byte_aligned())	
gci_alignment_zero_bit	f(1)
gci_num_reserved_bytes	u(8)
for(i = 0; i < gci_num_reserved_bytes; i++)	
gci_reserved_byte[i]	u(8)
}	

general_non_packed_constraint_flag equal to 1 specifies that there shall not be any frame packing arrangement SEI messages present in the bitstream of the OlsInScope.

general_non_packed_constraint_flag equal to 0 does not impose such a constraint.

NOTE 1 – Decoders may ignore the value of **general_non_packed_constraint_flag**, as there are no decoding process requirements associated with the presence or interpretation of frame packing arrangement SEI messages.

general_frame_only_constraint_flag equal to 1 specifies that OlsInScope conveys pictures that represent frames. **general_frame_only_constraint_flag** equal to 0 specifies that OlsInScope conveys pictures that may or may not represent frames.

NOTE 2 – Decoders may ignore the value of **general_frame_only_constraint_flag**, as there are no decoding process requirements associated with it.

general_non_projected_constraint_flag equal to 1 specifies that there shall not be any equirectangular projection SEI messages or generalized cubemap projection SEI messages present in the bitstream of the OlsInScope. **general_non_projected_constraint_flag** equal to 0 does not impose such a constraint.

NOTE 3 – Decoders may ignore the value of **general_non_projected_constraint_flag**, as there are no decoding process requirements associated with the presence or interpretation of equirectangular projection SEI messages and generalized cubemap projection SEI messages.

general_one_picture_only_constraint_flag equal to 1 specifies that there is only one coded picture in the bitstream. **general_one_picture_only_constraint_flag** equal to 0 does not impose such a constraint.

intra_only_constraint_flag equal to 1 specifies that **sh_slice_type** shall be equal to I.

intra_only_constraint_flag equal to 0 does not impose such a constraint.

max_bitdepth_minus8_constraint_idc specifies that **sps_bitdepth_minus8** shall be in the range of 0 to **max_bitdepth_minus8_constraint_idc**, inclusive.

max_chroma_format_constraint_idc specifies that `sps_chroma_format_idc` shall be in the range of 0 to `max_chroma_format_constraint_idc`, inclusive.

single_layer_constraint_flag equal to 1 specifies that the value of `nuh_layer_id` shall be the same for all VCL NAL units in `OlInScope`. `single_layer_constraint_flag` equal to 0 does not impose such a constraint.

all_layers_independent_constraint_flag equal to 1 specifies that `vps_all_independent_layers_flag` shall be equal to 1. `all_layers_independent_constraint_flag` equal to 0 does not impose such a constraint.

no_ref_pic_resampling_constraint_flag equal to 1 specifies that `sps_ref_pic_resampling_enabled_flag` shall be equal to 0. `no_ref_pic_resampling_constraint_flag` equal to 0 does not impose such a constraint.

no_res_change_in_clvs_constraint_flag equal to 1 specifies that `sps_res_change_in_clvs_allowed_flag` shall be equal to 0. `no_res_change_in_clvs_constraint_flag` equal to 0 does not impose such a constraint.

one_tile_per_pic_constraint_flag equal to 1 specifies that each picture shall contain only one tile, i.e., the value of `NumTilesInPic` for each picture shall be equal to 1. `one_tile_per_pic_constraint_flag` equal to 0 does not impose such a constraint.

pic_header_in_slice_header_constraint_flag equal to 1 specifies that each picture shall contain only one slice and the value of `sh_picture_header_in_slice_header_flag` in each slice shall be equal to 1.

`pic_header_in_slice_header_constraint_flag` equal to 0 does not impose such a constraint.

one_slice_per_pic_constraint_flag equal to 1 specifies that each picture shall contain only one slice, i.e., if `pps_rect_slice_flag` is equal to 1, the value of `num_slices_in_pic_minus1` shall be equal to 0, otherwise, the value of `num_tiles_in_slice_minus1` present in each slice header shall be equal to `NumTilesInPic - 1`.

`one_slice_per_pic_constraint_flag` equal to 0 does not impose such a constraint.

one_subpic_per_pic_constraint_flag equal to 1 specifies that each picture shall contain only one subpicture, i.e., the value of `sps_num_subpics_minus1` for each picture shall be equal to 0.

`one_subpic_per_pic_constraint_flag` equal to 0 does not impose such a constraint.

no_qtbt_dual_tree_intra_constraint_flag equal to 1 specifies that `sps_qtbt_dual_tree_intra_flag` shall be equal to 0. `no_qtbt_dual_tree_intra_constraint_flag` equal to 0 does not impose such a constraint.

no_partition_constraints_override_constraint_flag equal to 1 specifies that `sps_partition_constraints_override_enabled_flag` shall be equal to 0.

`no_partition_constraints_override_constraint_flag` equal to 0 does not impose such a constraint.

no_sao_constraint_flag equal to 1 specifies that `sps_sao_enabled_flag` shall be equal to 0.

`no_sao_constraint_flag` equal to 0 does not impose such a constraint.

no_alf_constraint_flag equal to 1 specifies that `sps_alf_enabled_flag` shall be equal to 0.

`no_alf_constraint_flag` equal to 0 does not impose such a constraint.

no_ccalf_constraint_flag equal to 1 specifies that **sps_ccalf_enabled_flag** shall be equal to 0.
no_ccalf_constraint_flag equal to 0 does not impose such a constraint.

no_joint_cbr_constraint_flag equal to 1 specifies that **sps_joint_cbr_enabled_flag** shall be equal to 0.
no_joint_cbr_constraint_flag equal to 0 does not impose such a constraint.

no_mrl_constraint_flag equal to 1 specifies that **sps_mrl_enabled_flag** shall be equal to 0.
no_mrl_constraint_flag equal to 0 does not impose such a constraint.

no_isp_constraint_flag equal to 1 specifies that **sps_isp_enabled_flag** shall be equal to 0.
no_isp_constraint_flag equal to 0 does not impose such a constraint.

no_mip_constraint_flag equal to 1 specifies that **sps_mip_enabled_flag** shall be equal to 0.
no_mip_constraint_flag equal to 0 does not impose such a constraint.

no_ref_wraparound_constraint_flag equal to 1 specifies that **sps_ref_wraparound_enabled_flag** shall be equal to 0. no_ref_wraparound_constraint_flag equal to 0 does not impose such a constraint.

no_temporal_mvp_constraint_flag equal to 1 specifies that **sps_temporal_mvp_enabled_flag** shall be equal to 0. no_temporal_mvp_constraint_flag equal to 0 does not impose such a constraint.

no_sbtmvp_constraint_flag equal to 1 specifies that **sps_sbtmvp_enabled_flag** shall be equal to 0.
no_sbtmvp_constraint_flag equal to 0 does not impose such a constraint.

no_amvr_constraint_flag equal to 1 specifies that **sps_amvr_enabled_flag** shall be equal to 0.
no_amvr_constraint_flag equal to 0 does not impose such a constraint.

no_bdof_constraint_flag equal to 1 specifies that **sps_bdof_enabled_flag** shall be equal to 0.
no_bdof_constraint_flag equal to 0 does not impose such a constraint.

no_dmvr_constraint_flag equal to 1 specifies that **sps_dmvr_enabled_flag** shall be equal to 0.
no_dmvr_constraint_flag equal to 0 does not impose such a constraint.

no_cclm_constraint_flag equal to 1 specifies that **sps_cclm_enabled_flag** shall be equal to 0.
no_cclm_constraint_flag equal to 0 does not impose such a constraint.

no_mts_constraint_flag equal to 1 specifies that **sps_mts_enabled_flag** shall be equal to 0.
no_mts_constraint_flag equal to 0 does not impose such a constraint.

no_sbt_constraint_flag equal to 1 specifies that **sps_sbt_enabled_flag** shall be equal to 0.
no_sbt_constraint_flag equal to 0 does not impose such a constraint.

no_lfnst_constraint_flag equal to 1 specifies that **sps_lfnst_enabled_flag** shall be equal to 0.
no_lfnst_constraint_flag equal to 0 does not impose such a constraint.

no_affine_motion_constraint_flag equal to 1 specifies that **sps_affine_enabled_flag** shall be equal to 0.
no_affine_motion_constraint_flag equal to 0 does not impose such a constraint.

no_mmvd_constraint_flag equal to 1 specifies that **sps_mmvd_enabled_flag** shall be equal to 0.
no_mmvd_constraint_flag equal to 0 does not impose such a constraint.

no_smvd_constraint_flag equal to 1 specifies that **sps_smvd_enabled_flag** shall be equal to 0.
no_smvd_constraint_flag equal to 0 does not impose such a constraint.

no_prof_constraint_flag equal to 1 specifies that **sps_affine_prof_enabled_flag** shall be equal to 0.
no_prof_constraint_flag equal to 0 does not impose such a constraint.

no_bcw_constraint_flag equal to 1 specifies that **sps_bcw_enabled_flag** shall be equal to 0.
no_bcw_constraint_flag equal to 0 does not impose such a constraint.

no_ibc_constraint_flag equal to 1 specifies that **sps_ibc_enabled_flag** shall be equal to 0.
no_ibc_constraint_flag equal to 0 does not impose such a constraint.

no_ciip_constraint_flag equal to 1 specifies that **sps_ciip_enabled_flag** shall be equal to 0.
no_cipp_constraint_flag equal to 0 does not impose such a constraint.

no_gpm_constraint_flag equal to 1 specifies that **sps_gpm_enabled_flag** shall be equal to 0.
no_gpm_constraint_flag equal to 0 does not impose such a constraint.

no_ladf_constraint_flag equal to 1 specifies that **sps_ladf_enabled_flag** shall be equal to 0.
no_ladf_constraint_flag equal to 0 does not impose such a constraint.

no_transform_skip_constraint_flag equal to 1 specifies that **sps_transform_skip_enabled_flag** shall be equal to 0. **no_transform_skip_constraint_flag** equal to 0 does not impose such a constraint.

no_bdpcm_constraint_flag equal to 1 specifies that **sps_bdpcm_enabled_flag** shall be equal to 0.
no_bdpcm_constraint_flag equal to 0 does not impose such a constraint.

no_weighted_prediction_constraint_flag equal to 1 specifies that **sps_weighted_pred_flag** and **sps_weighted_bipred_flag** shall both be equal to 0. **no_weighted_prediction_constraint_flag** equal to 0 does not impose such a constraint.

no_palette_constraint_flag equal to 1 specifies that **sps_palette_enabled_flag** shall be equal to 0.
no_palette_constraint_flag equal to 0 does not impose such a constraint.

no_act_constraint_flag equal to 1 specifies that **sps_act_enabled_flag** shall be equal to 0.
no_act_constraint_flag equal to 0 does not impose such a constraint.

no_lmcs_constraint_flag equal to 1 specifies that **sps_lmcs_enabled_flag** shall be equal to 0.
no_lmcs_constraint_flag equal to 0 does not impose such a constraint.

no_cu_qp_delta_constraint_flag equal to 1 specifies that **pps_cu_qp_delta_enabled_flag** shall be equal to 0. **no_cu_qp_delta_constraint_flag** equal to 0 does not impose such a constraint.

no_chroma_qp_offset_constraint_flag equal to 1 specifies that

pps_cu_chroma_qp_offset_list_enabled_flag shall be equal to 0. no_chroma_qp_offset_constraint_flag equal to 0 does not impose such a constraint.

no_dep_quant_constraint_flag equal to 1 specifies that sps_dep_quant_enabled_flag shall be equal to 0. no_dep_quant_constraint_flag equal to 0 does not impose such a constraint.

no_sign_data_hiding_constraint_flag equal to 1 specifies that sps_sign_data_hiding_enabled_flag shall be equal to 0. no_sign_data_hiding_constraint_flag equal to 0 does not impose such a constraint.

no_mixed_nalu_types_in_pic_constraint_flag equal to 1 specifies that it is a requirement of bitstream conformance that pps_mixed_nalu_types_in_pic_flag shall be equal to 0.

no_mixed_nalu_types_in_pic_constraint_flag equal to 0 does not impose such a constraint.

no_trail_constraint_flag equal to 1 specifies that there shall be no NAL unit with nuh_unit_type equal to TRAIL_NUT present in OlsInScope. no_trail_constraint_flag equal to 0 does not impose such a constraint.

no_stsa_constraint_flag equal to 1 specifies that there shall be no NAL unit with nuh_unit_type equal to STSA_NUT present in OlsInScope. no_stsa_constraint_flag equal to 0 does not impose such a constraint.

no_rasl_constraint_flag equal to 1 specifies that there shall be no NAL unit with nuh_unit_type equal to RASL_NUT present in OlsInScope. no_rasl_constraint_flag equal to 0 does not impose such a constraint.

no_radl_constraint_flag equal to 1 specifies that there shall be no NAL unit with nuh_unit_type equal to RADL_NUT present in OlsInScope. no_radl_constraint_flag equal to 0 does not impose such a constraint.

no_idr_constraint_flag equal to 1 specifies that there shall be no NAL unit with nuh_unit_type equal to IDR_W_RADL or IDR_N_LP present in OlsInScope. no_idr_constraint_flag equal to 0 does not impose such a constraint.

no_cra_constraint_flag equal to 1 specifies that there shall be no NAL unit with nuh_unit_type equal to CRA_NUT present in OlsInScope. no_cra_constraint_flag equal to 0 does not impose such a constraint.

no_gdr_constraint_flag equal to 1 specifies that sps_gdr_enabled_flag shall be equal to 0.

no_gdr_constraint_flag equal to 0 does not impose such a constraint.

no_aps_constraint_flag equal to 1 specifies that there shall be no NAL unit with nuh_unit_type equal to PREFIX_APS_NUT or SUFFIX_APS_NUT present in OlsInScope, and sps_lmcs_enabled_flag and sps_scaling_list_enabled_flag shall both be equal to 0. no_aps_constraint_flag equal to 0 does not impose such a constraint.

gci_alignment_zero_bits shall be equal to 0.

gci_num_reserved_bytes specifies the number of the reserved constraint bytes. The value of **gci_num_reserved_bytes** shall be 0. Other values of **gci_num_reserved_bytes** are reserved for future use by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. **gci_reserved_byte[i]** may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore the values of all the **gci_reserved_byte[i]** syntax elements.

3.6. Conditional signaling of GCI fields

[0039] JVET-S0050 and JVET-S0127 both proposes to add a presence flag, in the PTL syntax structure, to specifies the presence of the GCI syntax structure in the in the PTL syntax structure, with the only difference being that the following is done in S0050 and not done in S0127: S0050 adds byte alignment in the PTL syntax structure immediately after the GCI syntax structure, when present, to make sure that **general_level_idc** starts at a byte-aligned position in the PTL syntax structure, and consequently removed the byte alignment inside the GCI syntax structure. The JVET-S0050 syntax changes are as follows, wherein parts that have been added or modified are bolded, underlined and italicized, e.g., “using ***A and B***”, and some of the deleted parts are italicized with strikethrough, e.g., “based on ~~*A and B*~~”.

profile_tier_level(profileTierPresentFlag, maxNumSubLayersMinus1) {	Descriptor
if(profileTierPresentFlag) {	
general_profile_idc	u(7)
general_tier_flag	u(1)
<i><u>gci_present_flag</u></i>	<i><u>u(1)</u></i>
<i><u>if(gci_present_flag)</u></i>	
general_constraint_info()	
<i><u>while(!byte_aligned())</u></i>	
<i><u>ptl_alignment_zero_bit</u></i>	<i><u>f(1)</u></i>
}	
general_level_idc	u(8)
...	

general_constraint_info() {	Descriptor
...	

while(!byte_aligned())	
gci_alignment_zero_bit	1
gci_num_reserved_bytes //The 2 changes of "byte" to "bit" are purely editorial	u(8)
for(i = 0; i < gci_num_reserved_bytes; i++)	
gci_reserved_byte[i]	u(8)
}	

The JVET-S0127 syntax changes are as follows:

profile_tier_level(profileTierPresentFlag, maxNumSubLayersMinus1) {	Descriptor
if(profileTierPresentFlag) {	
general_profile_idc	u(7)
general_tier_flag	u(1)
<u>general_constraint_info_present_flag</u>	<u>u(1)</u>
<u>if(general_constraint_info_present_flag)</u>	
general_constraint_info()	
}	
general_level_idc	u(8)
...	

[0040] JVET-S0092 only changes the GCI syntax structure itself. It moves the GCI extension length indicator (gci_num_reserved_bytes) from last to first (gci_num_constraint_bytes) in the GCI syntax structure to enable skip signaling of the GCI fields. The value of gci_num_reserved_bytes shall be equal to either 0 or 9.

[0041] The S0092 syntax changes are as follows:

general_constraint_info() {	Descriptor
<u>gci_num_constraint_bytes</u>	<u>u(8)</u>
<u>if(gci_num_constraint_bytes > 8)</u>	
<u>{</u>	
general_non_packed_constraint_flag	u(1)
general_frame_only_constraint_flag	u(1)
... /* 61 more syntax elements */	

no_aps_constraint_flag	u(1)
while(!byte_aligned())	
gci_alignment_zero_bit	f(1)
_I	
gci_num_reserved_bytes	u(8)
for(i = 0; i < gci_num_reserved constraint bytes - 9; i++)	
gci_reserved_byte[i]	u(8)
}	

[0042] Here are some comparisons regarding the different approaches:

- 1) Regarding JVET-S0050 and JVET-S0127, the difference is on whether to make sure that `general_level_idc` is at a byte-aligned position in the PTL syntax structure (including when the GCI presence flag is equal to 0). The DCI, VPS, and SPS syntax have all been designed to make sure the each PTL syntax structure, when present, is at a byte-aligned position in the DCI/VPS/SPS. I hope we don't have to argue a lot to conclude that we should keep `general_level_idc` at a byte-aligned position in the PTL syntax structure. If that can be achieved, then JVET-S0050 and JVET-S0127 are the same.
- 2) The difference between having a GCI presence flag in the PTL syntax structure (as in JVET-S0050 and JVET-S0127) and manipulating the syntax within the GCI syntax structure as follows. They both specify that, between `general_tier_flag` and `general_level_idc`, there is some GCI information in the PTL syntax structure. The GCI presence flag approach uses a flag to specify the presence of the GCI fields, while the JVET-S0092 approach uses an 8-bit size indicator to achieve that purpose. Counting the number of bits, the JVET-S0050 approach uses 1 bit for the flag plus 0 to 7 byte alignment bits, while the JVET-S0092 approach uses 8 bits for the size indicator plus 0 to 7 byte alignment bits. So essentially the difference is just 7 bits.
- 3) There is one semantics error in JVET-S0092: if the value of `gci_num_reserved_bytes` shall be equal to either 0 or 9, then there can never be a `gci_reserved_byte[ i ]` in the GCI syntax structure. Of course, that can be easily resolved by saying something like the value of `gci_num_reserved_bytes` shall be either equal to 0 or equal to or greater than 9.

4. Technical problems solved by technical solutions disclosed herein

[0043] The existing designs for signaling of the CGI fields need to spend more bits than needed.

5. Invention

[0044] To solve the above problems and some other problems not mentioned, methods as summarized below are disclosed. The items should be considered as examples to explain the general concepts and should not be interpreted in a narrow way. Furthermore, these items can be applied individually or combined in any manner.

1) To solve the first problem, regarding conditional signaling of the GCI fields in a PTL syntax structure, one or more of the following approaches are disclosed, e.g., as in the 1st embodiment:

- a. Instead of being immediately after `general_tier_flag`, move the `general_constraint_info()` syntax structure and/or other GCI related syntax elements (e.g., presence of GCI flags) after the indication of level information.
 - i. In one example, move the `general_constraint_info()` syntax structure in the `profile_tier_level()` syntax structure to be immediately after the `general_sub_profile_idc[i]` syntax element.
- b. Add a new syntax element (e.g., a 1-bit flag, named `gci_present_flag`) in the `profile_tier_level()` syntax structure, to condition the presence of the `general_constraint_info()` syntax structure.
 - i. When `gci_present_flag` equal to 1 for a `profile_tier_level()` syntax structure with `profileTierPresentFlag` equal to 1, the `general_constraint_info()` syntax structure is present in the `profile_tier_level()` syntax structure. When `gci_present_flag` equal to 0 for a `profile_tier_level()` syntax structure (regardless of whether the `profileTierPresentFlag` equal to 1), the `general_constraint_info()` syntax structure is not present in the `profile_tier_level()` syntax structure.
 - ii. The semantics of is changed such that the semantics of the GCI fields only apply when they are present (i.e., when `gci_present_flag` is equal to 1). In other words, when `gci_present_flag` is equal to 0, no general constraints apply in addition to other specified constraints, such as those specified as part of the profile definition.
 - iii. Alternatively, furthermore, the new syntax element may be conditionally signalled, e.g., according to the value of `profileTierPresentFlag`.

- c. Remove the byte alignment syntax (i.e., the `gci_alignment_zero_bit` field and its syntax condition) from the `general_constraint_info()` syntax structure.
 - d. Instead of signaling the total number of reserved constraint bytes and the reserved constraint bytes, the number of reserved constraint bits and/or the value of each reserved constraint bit may be signalled.
 - i. In one example, change the GCI syntax element `gci_num_reserved_bytes` to `gci_num_reserved_bits`.
 - ii. Alternatively, furthermore, the changed GCI syntax element may be coded with `u(11)` instead of `u(8)`,
 - iii. Alternatively, furthermore, change the GCI syntax element `gci_reserved_byte[ i ]` to `gci_reserved_bit[ i ]`, coded with `u(1)` instead of `u(8)`.
- 2) To solve the first problem, regarding conditional signaling of the GCI fields in a PTL syntax structure, one or more of the following approaches are disclosed, e.g., as in the 2nd embodiment:
- a. Instead of being immediately after `general_tier_flag`, move the `general_constraint_info()` syntax structure and/or other GCI related syntax elements (e.g., presence of GCI flags) after the indication of level information.
 - i. In one example, move the `general_constraint_info()` syntax structure in the `profile_tier_level()` syntax structure to be immediately after the `general_sub_profile_idc[ i ]` syntax element (instead of being immediately after `general_tier_flag`).
 - b. Add a new syntax element (e.g., a 1-bit flag, named `gci_present_flag`), in the `profile_tier_level()` syntax structure, to condition the presence of the `general_constraint_info()` syntax structure.
 - i. When `gci_present_flag` equal to 1 for a `profile_tier_level()` syntax structure with `profileTierPresentFlag` equal to 1, the `general_constraint_info()` syntax structure is present in the `profile_tier_level()` syntax structure. When `gci_present_flag` equal to 0 for a `profile_tier_level()` syntax structure (regardless of whether the `profileTierPresentFlag` equal to 1), the `general_constraint_info()` syntax structure is not present in the `profile_tier_level()` syntax structure.

- ii. The semantics of is changed such that the semantics of the GCI fields only apply when they are present (i.e., when `gci_present_flag` is equal to 1). In other words, when `gci_present_flag` is equal to 0, no general constraints apply in addition to other specified constraints, such as those specified as part of the profile definition.
 - iii. Alternatively, furthermore, the new syntax element may be conditionally signalled, e.g., according to the value of `profileTierPresentFlag`.
- 3) To solve the first problem, regarding conditional signaling of the GCI fields in a PTL syntax structure, one or more of the following approaches are disclosed, e.g., as in the 1st embodiment:
 - a. Instead of being immediately after `general_tier_flag`, move the `general_constraint_info()` syntax structure and/or other GCI related syntax elements (e.g., presence of GCI flags) right before the byte alignment checking conditions in the PTL syntax structure (e.g., right before the `while( !byte_aligned() )` loop).
 - i. In one example, move the `general_constraint_info()` syntax structure in the `profile_tier_level()` syntax structure to be immediately after the `ptl_sublayer_level_present_flag[ i ]` syntax element (instead of being immediately after `general_tier_flag`).
 - b. Add a new syntax element (e.g., a 1-bit flag, named `gci_present_flag`), in the `profile_tier_level()` syntax structure, to condition the presence of the `general_constraint_info()` syntax structure.
 - i. When `gci_present_flag` equal to 1 for a `profile_tier_level()` syntax structure with `profileTierPresentFlag` equal to 1, the `general_constraint_info()` syntax structure is present in the `profile_tier_level()` syntax structure. When `gci_present_flag` equal to 0 for a `profile_tier_level()` syntax structure (regardless of whether the `profileTierPresentFlag` equal to 1), the `general_constraint_info()` syntax structure is not present in the `profile_tier_level()` syntax structure.
 - ii. The semantics of is changed such that the semantics of the GCI fields only apply when they are present (i.e., when `gci_present_flag` is equal to 1). In other words, when `gci_present_flag` is equal to 0, no general constraints apply

in addition to other specified constraints, such as those specified as part of the profile definition.

- iii. Alternatively, furthermore, the new syntax element may be conditionally signalled, e.g., according to the value of profileTierPresentFlag.
 - c. Remove the byte alignment syntax (i.e., the gci_alignment_zero_bit field and its syntax condition) from the general_constraint_info() syntax structure.
 - d. Instead of signaling the total number of reserved constraint bytes and the reserved constraint bytes, the number of reserved constraint bits and/or the value of each reserved constraint bit may be signalled.
 - i. In one example, change the GCI syntax elements gci_num_reserved_bytes to gci_num_reserved_bits
 - ii. Alternatively, furthermore, the changed GCI syntax element may be coded with u(11) instead of u(8).
 - iii. Alternatively, furthermore, change the GCI syntax element gci_reserved_byte[i] to gci_reserved_bit[i], coded with u(1) instead of u(8).
- 4) To solve the first problem, regarding conditional signaling of the GCI fields in a PTL syntax structure, one or more of the following approaches are disclosed, e.g., as in the 2nd embodiment:
- a. Instead of being immediately after general_tier_flag, move the general_constraint_info() syntax structure and/or other GCI related syntax elements (e.g., presence of GCI flags) right before the byte alignment checking conditions in the PTL syntax structure (e.g., right before the while(!byte_aligned()) loop).
 - i. In one example, move the general_constraint_info() syntax structure in the profile_tier_level() syntax structure to be immediately after the ptl_sublayer_level_present_flag[i] syntax element.
 - b. Add a new syntax element (e.g., a 1-bit flag, named gci_present_flag), in the profile_tier_level() syntax structure, to condition the presence of the general_constraint_info() syntax structure.
 - i. When gci_present_flag equal to 1 for a profile_tier_level() syntax structure with profileTierPresentFlag equal to 1, the general_constraint_info() syntax structure is present in the profile_tier_level() syntax structure. When

`gci_present_flag` equal to 0 for a `profile_tier_level()` syntax structure (regardless of whether the `profileTierPresentFlag` equal to 1), the `general_constraint_info()` syntax structure is not present in the `profile_tier_level()` syntax structure.

- ii. The semantics of is changed such that the semantics of the GCI fields only apply when they are present (i.e., when `gci_present_flag` is equal to 1). In other words, when `gci_present_flag` is equal to 0, no general constraints apply in addition to other specified constraints, such as those specified as part of the profile definition.
 - iii. Alternatively, furthermore, the new syntax element may be conditionally signalled, e.g., according to the value of `profileTierPresentFlag`.
- 5) To solve the first problem, regarding conditional signaling of the GCI fields in a PTL syntax structure, one or more of the following approaches are disclosed, e.g., as in the 5th embodiment:
- a. Instead of being before the `general_level_idc`, move the `general_constraint_info()` syntax structure and/or other GCI related syntax elements (e.g., presence of GCI flags) after the `general_level_idc`.
 - i. Move the `general_constraint_info()` syntax structure in the `profile_tier_level()` syntax structure to be immediately after the `general_level_idc` syntax element.
 - b. Move the byte alignment syntax (i.e., the `gci_alignment_zero_bit` field and its syntax condition) to the end of the `general_constraint_info()` syntax structure, i.e., after the GCI reserved fields instead of before the GCI reserved fields.
 - c. Instead of signaling the total number of reserved constraint bytes and the reserved constraint bytes, the number of reserved constraint bits and/or value of each reserved constraint bit may be signalled.
 - i. In one example, change the GCI syntax elements `gci_num_reserved_bytes` to `gci_num_reserved_bits`
 - ii. Alternatively, furthermore, the changed GCI syntax element may be coded with `u(11)` instead of `u(8)`, \

- iii. Alternatively, furthermore, change the GCI syntax element `gci_reserved_byte[ i ]` to `gci_reserved_bit[ i ]`, coded with `u(1)` instead of `u(8)`.
 - d. Add a new syntax element (e.g., a 1-bit flag, named `gci_present_flag`), at the beginning of the `general_constraint_info( )` syntax structure.
 - i. When `gci_present_flag` equal to 0, all fields in the `general_constraint_info( )` syntax structure, except the byte alignment fields, are skipped. The semantics of all these skipped fields are changed such that the semantics only apply when they are present (i.e., when `gci_present_flag` is equal to 1). In other words, when `gci_present_flag` is equal to 0, no general constraints apply in addition to other specified constraints, such as those specified as part of the profile definition.
- 6) To solve the first problem, regarding conditional signaling of the GCI fields in a PTL syntax structure, one or more of the following approaches are disclosed, e.g., as in the 6th embodiment:
 - a. Instead of being before the `general_level_idc`, move the `general_constraint_info( )` syntax structure and/or other GCI related syntax elements (e.g., presence of GCI flags) after the `general_level_idc`.
 - i. In one example, move the `general_constraint_info( )` syntax structure in the `profile_tier_level( )` syntax structure to be immediately after the `general_level_idc` syntax element.
 - b. Add a new syntax element (e.g., a 1-bit flag, named `gci_present_flag`), in the `profile_tier_level( )` syntax structure to condition the presence of `general_constraint_info( )` syntax structure.
 - i. When `gci_present_flag` equal to 1 for a `profile_tier_level( )` syntax structure with `profileTierPresentFlag` equal to 1, the `general_constraint_info( )` syntax structure is present in the `profile_tier_level( )` syntax structure. When `gci_present_flag` equal to 0 for a `profile_tier_level( )` syntax structure (regardless of whether the `profileTierPresentFlag` equal to 1), the `general_constraint_info( )` syntax structure is not present in the `profile_tier_level( )` syntax structure.

- ii. The semantics of all the GCI syntax structure is changed such that the semantics of the GCI fields only apply when they are present (i.e., when `gci_present_flag` is equal to 1). In other words, when `gci_present_flag` is equal to 0, no general constraints apply in addition to other specified constraints, such as those specified as part of the profile definition.
 - iii. To keep the starting position of `ptl_num_sub_profiles`, when present, to be a byte-aligned position, immediately after the `general_constraint_info()` syntax structure, add byte alignment check and, if not byte aligned, add `ptl_alignment_zero_bit` until it is byte aligned.
 - c. Remove the byte alignment syntax (i.e., the `gci_alignment_zero_bit` field and its syntax condition) from the `general_constraint_info()` syntax structure.
 - d. Instead of signaling the total number of reserved constraint bytes and the reserved constraint bytes, the number of reserved constraint bits and/or the value of each reserved constraint bit may be signalled.
 - i. In one example, change the GCI syntax element `gci_num_reserved_bytes` to `gci_num_reserved_bits`.
 - ii. Alternatively, furthermore, the changed syntax element may be coded with `u(11)` instead of `u(8)`,
 - iii. alternatively, furthermore, change the GCI syntax element `gci_reserved_byte[ i ]` to `gci_reserved_bit[ i ]`, coded with `u(1)` instead of `u(8)`.
- 7) To solve the first problem, regarding conditional signaling of the GCI fields in a PTL syntax structure, one or more of the following approaches are disclosed, e.g., as in the 5th embodiment:
- a. Instead of being before the `general_level_idc`, move the `general_constraint_info()` syntax structure and/or other GCI related syntax elements (e.g., presence of GCI flags) after the `general_level_idc`.
 - i. In one example, move the `general_constraint_info()` syntax structure in the `profile_tier_level()` syntax structure to be immediately after the `general_level_idc` syntax element.

- b. Add a new syntax element (e.g., a 1-bit flag, named `gci_present_flag`), in the `profile_tier_level()` syntax structure to condition the presence of `general_constraint_info()` syntax structure.
 - i. When `gci_present_flag` equal to 1 for a `profile_tier_level()` syntax structure with `profileTierPresentFlag` equal to 1, the `general_constraint_info()` syntax structure is present in the `profile_tier_level()` syntax structure. When `gci_present_flag` equal to 0 for a `profile_tier_level()` syntax structure (regardless of whether the `profileTierPresentFlag` equal to 1), the `general_constraint_info()` syntax structure is not present in the `profile_tier_level()` syntax structure.
 - ii. The semantics of all the GCI syntax structure is changed such that the semantics of the GCI fields only apply when they are present (i.e., when `gci_present_flag` is equal to 1). In other words, when `gci_present_flag` is equal to 0, no general constraints apply in addition to other specified constraints, such as those specified as part of the profile definition.
 - iii. To keep the starting position of `ptl_num_sub_profiles`, when present, to be a byte-aligned position, immediately after the `general_constraint_info()` syntax structure, add byte alignment check and, if not byte aligned, add `ptl_alignment_zero_bit` until it is byte aligned.
- 8) Change the semantics of `max_bitdepth_minus8_constraint_idc` to be as follows:
- `max_bitdepth_minus8_constraint_idc`** less than 8 specifies that `sps_bitdepth_minus8` shall be in the range of 0 to `max_bitdepth_minus8_constraint_idc`, inclusive.
- `max_bitdepth_minus8_constraint_idc` equal to or greater than 8 does not impose a constraint.
- a. Alternatively, "does not impose a constraint" above is changed to be "does not impose such a constraint".
- 9) Change the semantics of `max_chroma_format_constraint_idc` to be as follows:
- `max_chroma_format_constraint_idc`** less than 2 specifies that `sps_chroma_format_idc` shall be in the range of 0 to `max_chroma_format_constraint_idc`, inclusive.
- `max_chroma_format_constraint_idc` equal to 2 does not impose a constraint.
- a. Alternatively, "does not impose a constraint" above is changed to be "does not impose such a constraint".

6. Embodiments

[0045] Below are some example embodiments for some of the invention aspects summarized above in this Section, which can be applied to the VVC specification. The changed texts are based on the latest VVC text in JVET-S0152-v3. Most relevant parts that have been added or modified are bolded, underlined and italicized, e.g., “using A **and B**”, and some of the deleted parts are italicized with strikethrough, e.g., “based on *A and B*”.

6.1. Embodiment 1

This embodiment is for item 1 and its sub-items.

The syntax structure `profile_tier_level ( )` is changed as follows:

<code>profile_tier_level(profileTierPresentFlag, maxNumSubLayersMinus1) {</code>	Descriptor
<code>if(profileTierPresentFlag) {</code>	
<code> general_profile_idc</code>	<code>u(7)</code>
<code> general_tier_flag</code>	<code>u(1)</code>
<code> <i>general_constraint_info()</i></code>	
<code>}</code>	
<code> general_level_idc</code>	<code>u(8)</code>
<code>if(profileTierPresentFlag) {</code>	
<code> ptl_num_sub_profiles</code>	<code>u(8)</code>
<code> for(i = 0; i < ptl_num_sub_profiles; i++)</code>	
<code> general_sub_profile_idc[i]</code>	<code>u(32)</code>
<code> <i>gci_present_flag</i></code>	<u><code>u(1)</code></u>
<code> <i>if(gci_present_flag)</i></code>	
<code> <i>general_constraint_info()</i></code>	
<code> }</code>	
<code> for(i = 0; i < maxNumSubLayersMinus1; i++)</code>	
<code> ptl_sublayer_level_present_flag[i]</code>	<code>u(1)</code>
<code> while(!byte_aligned())</code>	
<code> ptl_alignment_zero_bit</code>	<code>f(1)</code>
<code> for(i = 0; i < maxNumSubLayersMinus1; i++)</code>	
<code> if(ptl_sublayer_level_present_flag[i])</code>	
<code> sublayer_level_idc[i]</code>	<code>u(8)</code>
<code> }</code>	

And the semantics are changed as follows:

...

gci_present_flag equal to 1 specifies that the general_constraint_info() syntax structure is present in the profile_tier_level() syntax structure when profileTierPresentFlag is equal to 1. gci_present_flag

equal to 0 specifies that the general_constraint_info() syntax structure is not present in the profile_tier_level() syntax structure.

The semantics of the GCI fields in general_constraint_info() syntax structure apply when gci_present_flag is equal to 1. When gci_present_flag is equal to 0, the general_constraint_info() syntax structure does not impose any constraint.

...

The syntax structure general_constraint_info() is changed as follows:

general_constraint_info() {	Descriptor
...	
while(!byte_aligned())	
gei_alignment_zero_bit	f(1)
gci_num_reserved_bits	u(8<u>11</u>)
for(i = 0; i < gci_num_reserved_bits; i++)	
gci_reserved_bit[i]	u(8<u>1</u>)
}	

And the semantics are changed as follows:

...

...

~~gei_alignment_zero_bits shall be equal to 0.~~

gci_num_reserved_bits specifies the number of the reserved constraint ~~bits~~. The value of gci_num_reserved_bits shall be 0. Other values of gci_num_reserved_bits are reserved for future use by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification.

gci_reserved_bit[i] may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore the values of all the gci_reserved_bit[i] syntax elements.

6.2. Embodiment 2

This embodiment is for item 2 and its sub-items.

The syntax structure profile_tier_level() is changed as follows:

profile_tier_level(profileTierPresentFlag, maxNumSubLayersMinus1) {	Descriptor
if(profileTierPresentFlag) {	
general_profile_idc	u(7)
general_tier_flag	u(1)
general_constraint_info()	

}	
general_level_idc	u(8)
if(profileTierPresentFlag) {	
ptl_num_sub_profiles	u(8)
for(i = 0; i < ptl_num_sub_profiles; i++)	
general_sub_profile_idc[i]	u(32)
<u>gci_present_flag</u>	<u>u(1)</u>
<u>if(gci_present_flag)</u>	
<u>general_constraint_info()</u>	
}	
for(i = 0; i < maxNumSubLayersMinus1; i++)	
ptl_sublayer_level_present_flag[i]	u(1)
while(!byte_aligned())	
ptl_alignment_zero_bit	f(1)
for(i = 0; i < maxNumSubLayersMinus1; i++)	
if(ptl_sublayer_level_present_flag[i])	
sublayer_level_idc[i]	u(8)
}	

And the semantics are changed as follows:

...

gci_present_flag equal to 1 specifies that the general_constraint_info() syntax structure is present in the profile_tier_level() syntax structure when profileTierPresentFlag is equal to 1. gci_present_flag equal to 0 specifies that the general_constraint_info() syntax structure is not present in the profile_tier_level() syntax structure.

The semantics of the GCI fields in general_constraint_info() syntax structure apply when gci_present_flag is equal to 1. When gci_present_flag is equal to 0, the general_constraint_info() syntax structure does not impose any constraint.

...

6.3. Embodiment 3

This embodiment is for item 3 and its sub-items.

The syntax structure profile_tier_level () is changed as follows:

profile_tier_level(profileTierPresentFlag, maxNumSubLayersMinus1) {	Descriptor
if(profileTierPresentFlag) {	
general_profile_idc	u(7)
general_tier_flag	u(1)
general_constraint_info()	

}	
general_level_idc	u(8)
if(profileTierPresentFlag) {	
ptl_num_sub_profiles	u(8)
for(i = 0; i < ptl_num_sub_profiles; i++)	
general_sub_profile_idc[i]	u(32)
}	
for(i = 0; i < maxNumSubLayersMinus1; i++)	
ptl_sublayer_level_present_flag[i]	u(1)
<u>if(profileTierPresentFlag) {</u>	
<u>gci_present_flag</u>	<u>u(1)</u>
<u>if(gci_present_flag)</u>	
<u>general_constraint_info()</u>	
<u>}</u>	
while(!byte_aligned())	
ptl_alignment_zero_bit	f(1)
for(i = 0; i < maxNumSubLayersMinus1; i++)	
if(ptl_sublayer_level_present_flag[i])	
sublayer_level_idc[i]	u(8)
}	

And the semantics are changed as follows:

...

gci_present_flag equal to 1 specifies that the general_constraint_info() syntax structure is present in the profile_tier_level() syntax structure when profileTierPresentFlag is equal to 1. gci_present_flag equal to 0 specifies that the general_constraint_info() syntax structure is not present in the profile_tier_level() syntax structure.

The semantics of the GCI fields in general_constraint_info() syntax structure apply when gci_present_flag is equal to 1. When gci_present_flag is equal to 0, the general_constraint_info() syntax structure does not impose any constraint.

...

In one example, alternatively, furthermore, the syntax structure general_constraint_info() is changed as follows:

general_constraint_info() {	Descriptor
...	
while(!byte_aligned())	
gci_alignment_zero_bit	f(1)

gci_num_reserved_bytes	u(811)
for(i = 0; i < gci_num_reserved_bytes; i++)	
gci_reserved_byte[i]	u(81)
}	

And the semantics are changed as follows:

...

~~gci_alignment_zero_bits shall be equal to 0.~~

gci_num_reserved_bytes specifies the number of the reserved constraint **bytes**. The value of **gci_num_reserved_bytes** shall be 0. Other values of **gci_num_reserved_bytes** are reserved for future use by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification.

gci_reserved_byte[i] may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore the values of all the **gci_reserved_bytes[i]** syntax elements.

6.4. Embodiment 4

This embodiment is for item 4 and its sub-items.

The syntax structure **profile_tier_level ()** is changed as follows:

profile_tier_level(profileTierPresentFlag, maxNumSubLayersMinus1) {	Descriptor
if(profileTierPresentFlag) {	
general_profile_idc	u(7)
general_tier_flag	u(1)
general_constraint_info()	
}	
general_level_idc	u(8)
if(profileTierPresentFlag) {	
ptl_num_sub_profiles	u(8)
for(i = 0; i < ptl_num_sub_profiles; i++)	
general_sub_profile_idc[i]	u(32)
}	
for(i = 0; i < maxNumSubLayersMinus1; i++)	
ptl_sublayer_level_present_flag[i]	u(1)
if(profileTierPresentFlag) {	
gci_present_flag	u(1)
if(gci_present_flag)	
general_constraint_info()	
}	

while(!byte_aligned())	
ptl_alignment_zero_bit	f(1)
for(i = 0; i < maxNumSubLayersMinus1; i++)	
if(ptl_sublayer_level_present_flag[i])	
sublayer_level_idc[i]	u(8)
}	

And the semantics are changed as follows:

...

gci_present_flag equal to 1 specifies that the general_constraint_info() syntax structure is present in the profile_tier_level() syntax structure when profileTierPresentFlag is equal to 1. gci_present_flag equal to 0 specifies that the general_constraint_info() syntax structure is not present in the profile_tier_level() syntax structure.

The semantics of the GCI fields in general_constraint_info() syntax structure apply when gci_present_flag is equal to 1. When gci_present_flag is equal to 0, the general_constraint_info() syntax structure does not impose any constraint.

...

6.5. Embodiment 5

This embodiment is for item 5 and its sub-items.

The syntax structure profile_tier_level () is changed as follows:

profile_tier_level(profileTierPresentFlag, maxNumSubLayersMinus1) {	Descriptor
if(profileTierPresentFlag) {	
general_profile_idc	u(7)
general_tier_flag	u(1)
general_constraint_info()	
}	
general_level_idc	u(8)
<u><i>if(profileTierPresentFlag)</i></u>	
<u><i>general_constraint_info()</i></u>	
if(profileTierPresentFlag) {	
ptl_num_sub_profiles	u(8)
for(i = 0; i < ptl_num_sub_profiles; i++)	
general_sub_profile_idc[i]	u(32)
}	
for(i = 0; i < maxNumSubLayersMinus1; i++)	
ptl_sublayer_level_present_flag[i]	u(1)
while(!byte_aligned())	

ptl_alignment_zero_bit	f(1)
for(i = 0; i < maxNumSubLayersMinus1; i++)	
if(ptl_sublayer_level_present_flag[i])	
sublayer_level_idc[i]	u(8)
}	

The syntax structure `general_constraint_info()` is changed as follows:

<code>general_constraint_info()</code> {	Descriptor
<u><i>gci_present_flag</i></u>	<u><i>u(1)</i></u>
<u><i>if(gci_present_flag) {</i></u>	
general_non_packed_constraint_flag	u(1)
general_frame_only_constraint_flag	u(1)
general_non_projected_constraint_flag	u(1)
general_one_picture_only_constraint_flag	u(1)
intra_only_constraint_flag	u(1)
max_bitdepth_minus8_constraint_idc	u(4)
max_chroma_format_constraint_idc	u(2)
single_layer_constraint_flag	u(1)
all_layers_independent_constraint_flag	u(1)
no_ref_pic_resampling_constraint_flag	u(1)
no_res_change_in_clvs_constraint_flag	u(1)
one_tile_per_pic_constraint_flag	u(1)
pic_header_in_slice_header_constraint_flag	u(1)
one_slice_per_pic_constraint_flag	u(1)
one_subpic_per_pic_constraint_flag	u(1)
no_qtbt_dual_tree_intra_constraint_flag	u(1)
no_partition_constraints_override_constraint_flag	u(1)
no_sao_constraint_flag	u(1)
no_alf_constraint_flag	u(1)
no_ccalf_constraint_flag	u(1)
no_joint_cbr_constraint_flag	u(1)
no_mrl_constraint_flag	u(1)
no_isp_constraint_flag	u(1)
no_mip_constraint_flag	u(1)
no_ref_wraparound_constraint_flag	u(1)
no_temporal_mvp_constraint_flag	u(1)
no_sbtmvp_constraint_flag	u(1)
no_amvr_constraint_flag	u(1)
no_bdof_constraint_flag	u(1)

no_dmvr_constraint_flag	u(1)
no_cclm_constraint_flag	u(1)
no_mts_constraint_flag	u(1)
no_sbt_constraint_flag	u(1)
no_lfnst_constraint_flag	u(1)
no_affine_motion_constraint_flag	u(1)
no_mmvd_constraint_flag	u(1)
no_smvd_constraint_flag	u(1)
no_prof_constraint_flag	u(1)
no_bcw_constraint_flag	u(1)
no_ibc_constraint_flag	u(1)
no_ciip_constraint_flag	u(1)
no_gpm_constraint_flag	u(1)
no_ladf_constraint_flag	u(1)
no_transform_skip_constraint_flag	u(1)
no_bdpcm_constraint_flag	u(1)
no_weighted_pred_constraint_flag	u(1)
no_palette_constraint_flag	u(1)
no_act_constraint_flag	u(1)
no_lmcs_constraint_flag	u(1)
no_cu_qp_delta_constraint_flag	u(1)
no_chroma_qp_offset_constraint_flag	u(1)
no_dep_quant_constraint_flag	u(1)
no_sign_data_hiding_constraint_flag	u(1)
no_mixed_nalu_types_in_pic_constraint_flag	u(1)
no_trail_constraint_flag	u(1)
no_stsa_constraint_flag	u(1)
no_rasl_constraint_flag	u(1)
no_radl_constraint_flag	u(1)
no_idr_constraint_flag	u(1)
no_cra_constraint_flag	u(1)
no_gdr_constraint_flag	u(1)
no_aps_constraint_flag	u(1)
while(!byte_aligned())	
gei_alignment_zero_bit	1
gci_num_reserved_bits bytes	u(811)
for(i = 0; i < gci_num_reserved_bits bytes ; i++)	
gci_reserved_bit byte [i]	u(81)
}	

<u>while(!byte_aligned())</u>	
<u>gci_alignment_zero_bit</u>	<u>f(1)</u>
}	

And the semantics are changed as follows:

...

gci_present_flag equal to 1 specifies that GCI fields are present in the general_constraint_info() syntax structure. gci_present_flag equal to 0 specifies that GCI fields are not present in the general_constraint_info() syntax structure.

The semantics of the GCI fields specified below apply when gci_present_flag is equal to 1. When gci_present_flag is equal to 0, the general_constraint_info() syntax structure does not impose any constraint.

...

gci_alignment_zero_bits shall be equal to 0.

gci_num_reserved_bits specifies the number of the reserved constraint bits. The value of gci_num_reserved_bits shall be 0. Other values of gci_num_reserved_bits are reserved for future use by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification.

gci_reserved_bits[i] may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore the values of all the gci_reserved_bits[i] syntax elements.

gci_alignment_zero_bits shall be equal to 0.

6.6. Embodiment 6

This embodiment is for item 6 and its sub-items.

The syntax structure profile_tier_level () is changed as follows:

profile_tier_level(profileTierPresentFlag, maxNumSubLayersMinus1) {	Descriptor
if(profileTierPresentFlag) {	
general_profile_idc	u(7)
general_tier_flag	u(1)
general_constraint_info()	
}	
general_level_idc	u(8)
<u>if(profileTierPresentFlag) {</u>	
<u>gci_present_flag</u>	<u>u(1)</u>
<u>if(gci_present_flag)</u>	
<u>general_constraint_info()</u>	

<u>while(!byte_aligned())</u>	
<u>ptl_alignment_zero_bit</u>	<u>f(1)</u>
<u>}</u>	
if(profileTierPresentFlag) {	
ptl_num_sub_profiles	u(8)
for(i = 0; i < ptl_num_sub_profiles; i++)	
general_sub_profile_idc[i]	u(32)
}	
for(i = 0; i < maxNumSubLayersMinus1; i++)	
ptl_sublayer_level_present_flag[i]	u(1)
while(!byte_aligned())	
ptl_alignment_zero_bit	f(1)
for(i = 0; i < maxNumSubLayersMinus1; i++)	
if(ptl_sublayer_level_present_flag[i])	
sublayer_level_idc[i]	u(8)
}	

And the semantics are changed as follows:

...

gci_present_flag equal to 1 specifies that the general_constraint_info() syntax structure is present in the profile_tier_level() syntax structure when profileTierPresentFlag is equal to 1. gci_present_flag equal to 0 specifies that the general_constraint_info() syntax structure is not present in the profile_tier_level() syntax structure.

The semantics of the GCI fields in general_constraint_info() syntax structure apply when gci_present_flag is equal to 1. When gci_present_flag is equal to 0, the general_constraint_info() syntax structure does not impose any constraint.

ptl_alignment_zero_bit shall be equal to 0.

...

The syntax structure general_constraint_info () is changed as follows:

general_constraint_info () {	Descriptor
...	
while(!byte_aligned())	
gci_alignment_zero_bit	f(1)
gci_num_reserved_bits ^{bytes}	u(811)
for(i = 0; i < gci_num_reserved_ bits ^{bytes} ; i++)	
gci_reserved_ bits ^{byte} [i]	u(81)

}	
---	--

And the semantics are changed as follows:

...
...

gci_alignment_zero_bits shall be equal to 0.

gci_num_reserved_bits specifies the number of the reserved constraint **bits**. The value of **gci_num_reserved_bits** shall be 0. Other values of **gci_num_reserved_bits** are reserved for future use by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification.

gci_reserved_bits[i] may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore the values of all the **gci_reserved_bits**[i] syntax elements.

6.7. Embodiment 7

This embodiment is for item 7 and its sub-items.

The syntax structure **profile_tier_level ()** is changed as follows:

profile_tier_level(profileTierPresentFlag, maxNumSubLayersMinus1) {	Descriptor
if(profileTierPresentFlag) {	
general_profile_idc	u(7)
general_tier_flag	u(1)
general_constraint_info()	
}	
general_level_idc	u(8)
if(profileTierPresentFlag) {	
gci_present_flag	u(1)
if(gci_present_flag)	
general_constraint_info()	
while(!byte_aligned())	
ptl_alignment_zero_bit	f(1)
}	
if(profileTierPresentFlag) {	
ptl_num_sub_profiles	u(8)
for(i = 0; i < ptl_num_sub_profiles; i++)	
general_sub_profile_idc [i]	u(32)
}	
for(i = 0; i < maxNumSubLayersMinus1; i++)	
ptl_sublayer_level_present_flag [i]	u(1)

while(!byte_aligned())	
ptl_alignment_zero_bit	f(1)
for(i = 0; i < maxNumSubLayersMinus1; i++)	
if(ptl_sublayer_level_present_flag[i])	
sublayer_level_idc[i]	u(8)
}	

And the semantics are changed as follows:

...

gci_present_flag equal to 1 specifies that the general_constraint_info() syntax structure is present in the profile_tier_level() syntax structure when profileTierPresentFlag is equal to 1. gci_present_flag equal to 0 specifies that the general_constraint_info() syntax structure is not present in the profile_tier_level() syntax structure.

The semantics of the GCI fields in general_constraint_info() syntax structure apply when gci_present_flag is equal to 1. When gci_present_flag is equal to 0, the general_constraint_info() syntax structure does not impose any constraint.

ptl_alignment_zero_bit shall be equal to 0.

...

[0046] FIG. 1 is a block diagram showing an example video processing system 1000 in which various techniques disclosed herein may be implemented. Various implementations may include some or all of the components of the system 1000. The system 1000 may include input 1002 for receiving video content. The video content may be received in a raw or uncompressed format, e.g., 8 or 10 bit multi-component pixel values, or may be in a compressed or encoded format. The input 1002 may represent a network interface, a peripheral bus interface, or a storage interface. Examples of network interface include wired interfaces such as Ethernet, passive optical network (PON), etc. and wireless interfaces such as Wi-Fi or cellular interfaces.

[0047] The system 1000 may include a coding component 1004 that may implement the various coding or encoding methods described in the present document. The coding component 1004 may reduce the average bitrate of video from the input 1002 to the output of the coding component 1004 to produce a coded representation of the video. The coding techniques are therefore sometimes called video compression or video transcoding techniques. The output of the coding component 1004 may be either stored, or transmitted via a communication connected, as represented by the component 1006. The stored or communicated bitstream (or coded)

representation of the video received at the input 1002 may be used by the component 1008 for generating pixel values or displayable video that is sent to a display interface 1010. The process of generating user-viewable video from the bitstream representation is sometimes called video decompression. Furthermore, while certain video processing operations are referred to as “coding” operations or tools, it will be appreciated that the coding tools or operations are used at an encoder and corresponding decoding tools or operations that reverse the results of the coding will be performed by a decoder.

[0048] Examples of a peripheral bus interface or a display interface may include universal serial bus (USB) or high definition multimedia interface (HDMI) or Displayport, and so on. Examples of storage interfaces include SATA (serial advanced technology attachment), PCI, IDE interface, and the like. The techniques described in the present document may be embodied in various electronic devices such as mobile phones, laptops, smartphones or other devices that are capable of performing digital data processing and/or video display.

[0049] FIG. 2 is a block diagram of a video processing apparatus 2000. The apparatus 2000 may be used to implement one or more of the methods described herein. The apparatus 2000 may be embodied in a smartphone, tablet, computer, Internet of Things (IoT) receiver, and so on. The apparatus 2000 may include one or more processors 2002, one or more memories 2004 and video processing hardware 2006. The processor(s) 2002 may be configured to implement one or more methods described in the present document (e.g., in FIGS. 6-9). The memory (memories) 2004 may be used for storing data and code used for implementing the methods and techniques described herein. The video processing hardware 2006 may be used to implement, in hardware circuitry, some techniques described in the present document. In some embodiments, the hardware 2006 may be partly or entirely in the one or more processors 2002, e.g., a graphics processor.

[0050] FIG. 3 is a block diagram that illustrates an example video coding system 100 that may utilize the techniques of this disclosure. As shown in FIG. 3, video coding system 100 may include a source device 110 and a destination device 120. Source device 110 generates encoded video data which may be referred to as a video encoding device. Destination device 120 may decode the encoded video data generated by source device 110 which may be referred to as a video decoding device. Source device 110 may include a video source 112, a video encoder 114, and an input/output (I/O) interface 116.

[0051] Video source 112 may include a source such as a video capture device, an interface to receive video data from a video content provider, and/or a computer graphics system for generating video data, or a combination of such sources. The video data may comprise one or more pictures. Video encoder 114 encodes the video data from video source 112 to generate a bitstream. The bitstream may include a sequence of bits that form a coded representation of the video data. The bitstream may include coded pictures and associated data. The coded picture is a coded representation of a picture. The associated data may include sequence parameter sets, picture parameter sets, and other syntax structures. I/O interface 116 may include a modulator/demodulator (modem) and/or a transmitter. The encoded video data may be transmitted directly to destination device 120 via I/O interface 116 through network 130a. The encoded video data may also be stored onto a storage medium/server 130b for access by destination device 120.

[0052] Destination device 120 may include an I/O interface 126, a video decoder 124, and a display device 122.

[0053] I/O interface 126 may include a receiver and/or a modem. I/O interface 126 may acquire encoded video data from the source device 110 or the storage medium/ server 130b. Video decoder 124 may decode the encoded video data. Display device 122 may display the decoded video data to a user. Display device 122 may be integrated with the destination device 120, or may be external to destination device 120 which be configured to interface with an external display device.

[0054] Video encoder 114 and video decoder 124 may operate according to a video compression standard, such as the High Efficiency Video Coding (HEVC) standard, Versatile Video Coding (VVC) standard and other current and/or further standards.

[0055] FIG. 4 is a block diagram illustrating an example of video encoder 200, which may be video encoder 114 in the system 100 illustrated in FIG. 3.

[0056] Video encoder 200 may be configured to perform any or all of the techniques of this disclosure. In the example of FIG. 4, video encoder 200 includes a plurality of functional components. The techniques described in this disclosure may be shared among the various components of video encoder 200. In some examples, a processor may be configured to perform any or all of the techniques described in this disclosure.

[0057] The functional components of video encoder 200 may include a partition unit 201, a predication unit 202 which may include a mode select unit 203, a motion estimation unit 204, a motion compensation unit 205 and an intra prediction unit 206, a residual generation unit 207, a transform unit 208, a quantization unit 209, an inverse quantization unit 210, an inverse transform unit 211, a reconstruction unit 212, a buffer 213, and an entropy encoding unit 214.

[0058] In other examples, video encoder 200 may include more, fewer, or different functional components. In an example, predication unit 202 may include an intra block copy(IBC) unit. The IBC unit may perform predication in an IBC mode in which at least one reference picture is a picture where the current video block is located.

[0059] Furthermore, some components, such as motion estimation unit 204 and motion compensation unit 205 may be highly integrated, but are represented in the example of FIG. 4 separately for purposes of explanation.

[0060] Partition unit 201 may partition a picture into one or more video blocks. Video encoder 200 and video decoder 300 may support various video block sizes.

[0061] Mode select unit 203 may select one of the coding modes, intra or inter, e.g., based on error results, and provide the resulting intra- or inter-coded block to a residual generation unit 207 to generate residual block data and to a reconstruction unit 212 to reconstruct the encoded block for use as a reference picture. In some example, Mode select unit 203 may select a combination of intra and inter predication (CIIP) mode in which the predication is based on an inter predication signal and an intra predication signal. Mode select unit 203 may also select a resolution for a motion vector (e.g., a sub-pixel or integer pixel precision) for the block in the case of inter-predication.

[0062] To perform inter prediction on a current video block, motion estimation unit 204 may generate motion information for the current video block by comparing one or more reference frames from buffer 213 to the current video block. Motion compensation unit 205 may determine a predicted video block for the current video block based on the motion information and decoded samples of pictures from buffer 213 other than the picture associated with the current video block.

[0063] Motion estimation unit 204 and motion compensation unit 205 may perform different operations for a current video block, for example, depending on whether the current video block is in an I slice, a P slice, or a B slice.

[0064] In some examples, motion estimation unit 204 may perform uni-directional prediction for the current video block, and motion estimation unit 204 may search reference pictures of list 0 or list 1 for a reference video block for the current video block. Motion estimation unit 204 may then generate a reference index that indicates the reference picture in list 0 or list 1 that contains the reference video block and a motion vector that indicates a spatial displacement between the current video block and the reference video block. Motion estimation unit 204 may output the reference index, a prediction direction indicator, and the motion vector as the motion information of the current video block. Motion compensation unit 205 may generate the predicted video block of the current block based on the reference video block indicated by the motion information of the current video block.

[0065] In other examples, motion estimation unit 204 may perform bi-directional prediction for the current video block, motion estimation unit 204 may search the reference pictures in list 0 for a reference video block for the current video block and may also search the reference pictures in list 1 for another reference video block for the current video block. Motion estimation unit 204 may then generate reference indexes that indicate the reference pictures in list 0 and list 1 containing the reference video blocks and motion vectors that indicate spatial displacements between the reference video blocks and the current video block. Motion estimation unit 204 may output the reference indexes and the motion vectors of the current video block as the motion information of the current video block. Motion compensation unit 205 may generate the predicted video block of the current video block based on the reference video blocks indicated by the motion information of the current video block.

[0066] In some examples, motion estimation unit 204 may output a full set of motion information for decoding processing of a decoder.

[0067] In some examples, motion estimation unit 204 may do not output a full set of motion information for the current video. Rather, motion estimation unit 204 may signal the motion information of the current video block with reference to the motion information of another video block. For example, motion estimation unit 204 may determine that the motion information of the current video block is sufficiently similar to the motion information of a neighboring video block.

[0068] In one example, motion estimation unit 204 may indicate, in a syntax structure associated with the current video block, a value that indicates to the video decoder 300 that the current video block has the same motion information as the another video block.

[0069] In another example, motion estimation unit 204 may identify, in a syntax structure associated with the current video block, another video block and a motion vector difference (MVD). The motion vector difference indicates a difference between the motion vector of the current video block and the motion vector of the indicated video block. The video decoder 300 may use the motion vector of the indicated video block and the motion vector difference to determine the motion vector of the current video block.

[0070] As discussed above, video encoder 200 may predictively signal the motion vector. Two examples of predictive signaling techniques that may be implemented by video encoder 200 include advanced motion vector predication (AMVP) and merge mode signaling.

[0071] Intra prediction unit 206 may perform intra prediction on the current video block. When intra prediction unit 206 performs intra prediction on the current video block, intra prediction unit 206 may generate prediction data for the current video block based on decoded samples of other video blocks in the same picture. The prediction data for the current video block may include a predicted video block and various syntax elements.

[0072] Residual generation unit 207 may generate residual data for the current video block by subtracting (e.g., indicated by the minus sign) the predicted video block(s) of the current video block from the current video block. The residual data of the current video block may include residual video blocks that correspond to different sample components of the samples in the current video block.

[0073] In other examples, there may be no residual data for the current video block for the current video block, for example in a skip mode, and residual generation unit 207 may not perform the subtracting operation.

[0074] Transform processing unit 208 may generate one or more transform coefficient video blocks for the current video block by applying one or more transforms to a residual video block associated with the current video block.

[0075] After transform processing unit 208 generates a transform coefficient video block associated with the current video block, quantization unit 209 may quantize the transform

coefficient video block associated with the current video block based on one or more quantization parameter (QP) values associated with the current video block.

[0076] Inverse quantization unit 210 and inverse transform unit 211 may apply inverse quantization and inverse transforms to the transform coefficient video block, respectively, to reconstruct a residual video block from the transform coefficient video block. Reconstruction unit 212 may add the reconstructed residual video block to corresponding samples from one or more predicted video blocks generated by the predication unit 202 to produce a reconstructed video block associated with the current block for storage in the buffer 213.

[0077] After reconstruction unit 212 reconstructs the video block, loop filtering operation may be performed reduce video blocking artifacts in the video block.

[0078] Entropy encoding unit 214 may receive data from other functional components of the video encoder 200. When entropy encoding unit 214 receives the data, entropy encoding unit 214 may perform one or more entropy encoding operations to generate entropy encoded data and output a bitstream that includes the entropy encoded data.

[0079] FIG. 5 is a block diagram illustrating an example of video decoder 300 which may be video decoder 114 in the system 100 illustrated in FIG. 3.

[0080] The video decoder 300 may be configured to perform any or all of the techniques of this disclosure. In the example of FIG. 5, the video decoder 300 includes a plurality of functional components. The techniques described in this disclosure may be shared among the various components of the video decoder 300. In some examples, a processor may be configured to perform any or all of the techniques described in this disclosure.

[0081] In the example of FIG. 5, video decoder 300 includes an entropy decoding unit 301, a motion compensation unit 302, an intra prediction unit 303, an inverse quantization unit 304, an inverse transformation unit 305, and a reconstruction unit 306 and a buffer 307. Video decoder 300 may, in some examples, perform a decoding pass generally reciprocal to the encoding pass described with respect to video encoder 200 (FIG. 4).

[0082] Entropy decoding unit 301 may retrieve an encoded bitstream. The encoded bitstream may include entropy coded video data (e.g., encoded blocks of video data). Entropy decoding unit 301 may decode the entropy coded video data, and from the entropy decoded video data, motion compensation unit 302 may determine motion information including motion vectors, motion vector precision, reference picture list indexes, and other motion information. Motion

compensation unit 302 may, for example, determine such information by performing the AMVP and merge mode.

[0083] Motion compensation unit 302 may produce motion compensated blocks, possibly performing interpolation based on interpolation filters. Identifiers for interpolation filters to be used with sub-pixel precision may be included in the syntax elements.

[0084] Motion compensation unit 302 may use interpolation filters as used by video encoder 20 during encoding of the video block to calculate interpolated values for sub-integer pixels of a reference block. Motion compensation unit 302 may determine the interpolation filters used by video encoder 200 according to received syntax information and use the interpolation filters to produce predictive blocks.

[0085] Motion compensation unit 302 may use some of the syntax information to determine sizes of blocks used to encode frame(s) and/or slice(s) of the encoded video sequence, partition information that describes how each macroblock of a picture of the encoded video sequence is partitioned, modes indicating how each partition is encoded, one or more reference frames (and reference frame lists) for each inter-encoded block, and other information to decode the encoded video sequence.

[0086] Intra prediction unit 303 may use intra prediction modes for example received in the bitstream to form a prediction block from spatially adjacent blocks. Inverse quantization unit 303 inverse quantizes, i.e., de-quantizes, the quantized video block coefficients provided in the bitstream and decoded by entropy decoding unit 301. Inverse transform unit 303 applies an inverse transform.

[0087] Reconstruction unit 306 may sum the residual blocks with the corresponding prediction blocks generated by motion compensation unit 202 or intra-prediction unit 303 to form decoded blocks. If desired, a deblocking filter may also be applied to filter the decoded blocks in order to remove blockiness artifacts. The decoded video blocks are then stored in buffer 307, which provides reference blocks for subsequent motion compensation/intra prediction and also produces decoded video for presentation on a display device.

[0088] FIGS. 6-15 show example methods that can implement the technical solution described above in, for example, the embodiments shown in FIGS. 1-5.

[0089] FIG. 6 shows a flowchart for an example method 600 of video processing. The method 600 includes, at operation 610, performing a conversion between a video comprising one or more

pictures and a bitstream of the video according to a rule that specifies that a syntax structure in a profile-tier-level syntax structure is after a syntax element, the syntax structure comprising information related to general constraint information (GCI) for the bitstream, and the syntax element indicating a level to which an output layer set associated with the profile-tier-level syntax structure conforms.

[0090] FIG. 7 shows a flowchart for an example method 700 of video processing. The method 700 includes, at operation 710, performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule that specifies that a byte alignment syntax in a general constraint information (GCI) syntax structure is after one or more GCI reserved fields, the byte alignment syntax indicating whether a current position in the bitstream is an integer multiple of 8 bits from a position of a first bit in the bitstream, and the GCI syntax structure comprising GCI related syntax elements.

[0091] FIG. 8 shows a flowchart for an example method 800 of video processing. The method 800 includes, at operation 810, performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule that specifies that a syntax structure in a profile-tier-level syntax structure is after an indication of level information, the syntax structure comprising information related to general constraint information (GCI), and the indication of level information specifying an interoperability indicator.

[0092] FIG. 9 shows a flowchart for an example method 900 of video processing. The method 900 includes, at operation 910, performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule that specifies that a syntax element in a profile-tier-level syntax structure indicates whether a general constraint information (GCI) syntax structure is included in the profile-tier-level syntax structure.

[0093] FIG. 10 shows a flowchart for an example method 1000 of video processing. The method 1000 includes, at operation 1010, performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule that specifies that a byte alignment syntax is excluded from a general constraint information (GCI) syntax structure that is present in the profile-tier-level syntax structure, the byte alignment syntax indicating whether a current position in the bitstream is an integer multiple of 8 bits from a position of a first bit in the bitstream, and the GCI syntax structure comprising a GCI related syntax element.

[0094] FIG. 11 shows a flowchart for an example method 1100 of video processing. The method 1100 includes, at operation 1110, performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule that specifies that a general constraint information (GCI) syntax structure is immediately before a byte alignment checking condition in a profile-tier-level syntax structure, the GCI syntax structure comprising GCI related syntax elements, and the byte alignment checking condition checking whether a current position in the bitstream is an integer multiple of 8 bits from a position of a first bit in the bitstream.

[0095] FIG. 12 shows a flowchart for an example method 1200 of video processing. The method 1200 includes, at operation 1210, performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule that specifies that a number of a plurality of reserved constraint bits associated with a general constraint information (GCI) syntax element is included in the bitstream.

[0096] FIG. 13 shows a flowchart for an example method 1300 of video processing. The method 1300 includes, at operation 1310, performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule that specifies that a general constraint information (GCI) syntax element is included at a beginning of a GCI syntax structure, the GCI syntax element indicating whether one or more GCI syntax elements are included in the GCI syntax structure.

[0097] FIG. 14 shows a flowchart for an example method 1400 of video processing. The method 1400 includes, at operation 1410, performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule that specifies a constraint on a syntax element, the syntax element corresponding to a bit depth used for representing the video in the bitstream.

[0098] FIG. 15 shows a flowchart for an example method 1500 of video processing. The method 1500 includes, at operation 1510, performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule that specifies a constraint on a syntax element, the syntax element corresponding to a chroma format of the video.

[0099] The following solutions show example embodiments of techniques discussed in the previous section (e.g., items 1-9).

[00100] A listing of solutions preferred by some embodiments is provided next.

[00101] A1. A method of video processing, comprising performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies that a syntax structure in a profile-tier-level syntax structure is after a syntax element, wherein the syntax structure comprises information related to general constraint information (GCI) for the bitstream, and wherein the syntax element indicates a level to which an output layer set associated with the profile-tier-level syntax structure conforms.

[00102] A2. The method of solution A1, wherein the information related to the GCI indicates whether one or more GCI flags are indicated.

[00103] A3. The method of solution A1 or A2, wherein the syntax structure is immediately after the syntax element.

[00104] A4. The method of any of solutions A1 to A3, wherein the syntax structure is a `general_constraint_info( )` syntax structure and the syntax element is a `general_level_idc` syntax element.

[00105] A5. A method of video processing, comprising performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies that a byte alignment syntax in a general constraint information (GCI) syntax structure is after one or more GCI reserved fields, wherein the byte alignment syntax indicates whether a current position in the bitstream is an integer multiple of 8 bits from a position of a first bit in the bitstream, and wherein the GCI syntax structure comprises GCI related syntax elements.

[00106] A6. The method of solution A5, wherein the byte alignment syntax is at an end of the GCI syntax structure.

[00107] A7. The method of solution A5 or A6, wherein the byte alignment syntax comprises a `gci_alignment_zero_bit` field and its syntax condition, and the GCI syntax structure is a `general_constraint_info( )` syntax structure.

[00108] A8. A method of video processing, comprising performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies that a syntax structure in a profile-tier-level syntax structure is after an indication of level information, wherein the syntax structure comprises information related to general constraint information (GCI), and wherein the indication of level information specifies an interoperability indicator.

[00109] A9. The method of solution A8, wherein the syntax structure is a `general_constraint_info( )` syntax structure and the indication is a `general_sub_profile_idc[ i ]` syntax element.

[00110] A10. A method of video processing, comprising performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein rule specifies that a syntax element in a profile-tier-level syntax structure indicates whether a general constraint information (GCI) syntax structure is included in the profile-tier-level syntax structure.

[00111] A11. The method of solution A10, wherein the syntax element is a first flag.

[00112] A12. The method of solution A11, wherein the first flag equaling one and a second flag equaling one indicates that the GCI syntax structure is present in the profile-tier-level syntax structure.

[00113] A13. The method of solution A12, wherein the second flag is `profileTierPresentFlag`.

[00114] A14. The method of solution A11, wherein the first flag equaling zero indicates that the GCI syntax structure is not present in the profile-tier-level syntax structure.

[00115] A15. The method of solution A11, wherein the GCI syntax structure comprises one or more GCI fields, and wherein semantics of the one or more GCI fields only apply when the first flag is equal to one.

[00116] A16. The method of solution A11, wherein an inclusion of the first flag in the profile-tier-level syntax structure is based on a value of a second flag.

[00117] A17. The method of solution A16, wherein the second flag is `profileTierPresentFlag`.

[00118] A18. The method of any of solutions A11 to A17, wherein the first flag is `gci_present_flag`.

[00119] A19. A method of video processing, comprising performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies that a byte alignment syntax is excluded from a general constraint information (GCI) syntax structure that is present in the profile-tier-level syntax structure, wherein the byte alignment syntax indicates whether a current position in the bitstream is an integer multiple of 8 bits from a position of a first bit in the bitstream, and wherein the GCI syntax structure comprises a GCI related syntax element.

[00120] A20. The method of solution A19, wherein the byte alignment syntax comprises a `gci_alignment_zero_bit` field and its syntax condition, and the GCI syntax structure is a `general_constraint_info()` syntax structure.

[00121] A21. The method of solution A19, wherein the rule further specifies that a number of a plurality of reserved constraint bits associated with the GCI related syntax element is included in the bitstream.

[00122] A22. The method of solution A21, wherein the rule further specifies that a value of each of the plurality of reserved constraint bits is included in the bitstream.

[00123] A23. The method of solution A22, wherein the GCI syntax element is `gci_num_reserved_bits`.

[00124] A24. The method of solution A22, wherein the syntax element is coded as an unsigned 11-bit integer.

[00125] A25. The method of solution A22, wherein the value of each of the plurality of reserved constraint bits is coded as an unsigned 1-bit integer.

[00126] A26. A method of video processing, comprising performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies that a general constraint information (GCI) syntax structure is immediately before a byte alignment checking condition in a profile-tier-level syntax structure, wherein the GCI syntax structure comprises GCI related syntax elements, and wherein the byte alignment checking condition checks whether a current position in the bitstream is an integer multiple of 8 bits from a position of a first bit in the bitstream.

[00127] A27. The method of solution A26, wherein the GCI syntax structure is a `general_constraint_info()` syntax structure, and wherein the byte alignment checking condition is based on a `byte_aligned()` syntax element.

[00128] A28. The method of any of solutions A1 to A27, wherein the conversion comprises decoding the video from the bitstream.

[00129] A29. The method of any of solutions A1 to A27, wherein the conversion comprises encoding the video into the bitstream.

[00130] A30. A method of storing a bitstream representing a video to a computer-readable recording medium, comprising generating the bitstream from the video according to a method

described in any one or more of solutions A1 to A27; and storing the bitstream in the computer-readable recording medium.

[00131] A31. A video processing apparatus comprising a processor configured to implement a method recited in any one or more of solutions A1 to A30.

[00132] A32. A computer-readable medium having instructions stored thereon, the instructions, when executed, causing a processor to implement a method recited in one or more of solutions A1 to A30.

[00133] A33. A computer readable medium that stores the bitstream generated according to any one or more of solutions A1 to A30.

[00134] A34. A video processing apparatus for storing a bitstream, wherein the video processing apparatus is configured to implement a method recited in any one or more of solutions A1 to A30.

[00135] Another listing of solutions preferred by some embodiments is provided next.

[00136] B1. A method of video processing, comprising performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies that a number of a plurality of reserved constraint bits associated with a general constraint information (GCI) syntax element is included in the bitstream.

[00137] B2. The method of solution B1, wherein the rule further specifies that a value of each of the plurality of reserved constraint bits is included in the bitstream.

[00138] B3. The method of solution B2, wherein the syntax element is coded as an unsigned 11-bit integer.

[00139] B4. The method of solution B2, wherein the value of each of the plurality of reserved constraint bits is coded as an unsigned 1-bit integer.

[00140] B5. The method of solution B2, wherein the GCI syntax element is `gci_num_reserved_bits`.

[00141] B6. A method of video processing, comprising performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies that a general constraint information (GCI) syntax element is included at a beginning of a GCI syntax structure, wherein the GCI syntax element indicates whether one or more GCI syntax elements are included in the GCI syntax structure.

[00142] B7. The method of solution B6, wherein the GCI syntax element equaling zero indicates that the one or more GCI syntax elements are not included in the GCI syntax structure.

[00143] B8. The method of solution B7, wherein the one or more GCI syntax elements exclude a byte alignment field.

[00144] B9. The method of solution B7, wherein the GCI syntax structure does not impose a constraint.

[00145] B10. The method of solution B6, wherein a second syntax element is after the GCI syntax structure, wherein the second syntax element specifies a number of syntax elements, each of which specifies an interoperability indicator.

[00146] B11. The method of solution B10, wherein the second syntax element is immediately after the GCI syntax structure.

[00147] B12. The method of solution B10 or B11, wherein a byte alignment check for the second syntax element is added in response to the second syntax element being present in the bitstream.

[00148] B13. The method of solution B12, wherein one or more alignment bits are added in response to the second syntax element not being byte-aligned, and wherein adding the one or more alignment bits causes the second syntax element to be byte-aligned.

[00149] B14. The method of solution B13, wherein the one or more alignment bits comprise one or more `ptl_alignment_zero_bit`.

[00150] B15. The method of any of solutions B10 to B14, wherein the second syntax element is `ptl_num_sub_profiles`.

[00151] B16. The method of solution B6, wherein the GCI syntax element equaling one indicates that the one or more GCI syntax elements are included in the GCI syntax structure.

[00152] B17. The method of any of solutions B6 to B16, wherein the first GCI syntax element is `gci_present_flag` and the GCI syntax structure is a `general_constraint_info()` syntax structure.

[00153] B18. A method of video processing, comprising performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies a constraint on a syntax element, wherein the syntax element corresponds to a bit depth used for representing the video in the bitstream.

[00154] B19. The method of solution B18, wherein the constraint specifies that the syntax element is a non-negative integer less than a maximum value that equals a value of the syntax element plus one.

[00155] B20. The method of solution B19, wherein the syntax element being less than 8 imposes the constraint.

[00156] B21. The method of solution B19, wherein the syntax element being greater than or equal to 8 does not impose the constraint.

[00157] B22. The method of any of solutions B18 to B21, wherein the syntax element is `max_bitdepth_minus8_constraint_idc`.

[00158] B23. A method of video processing, comprising performing a conversion between a video comprising one or more pictures and a bitstream of the video according to a rule, wherein the rule specifies a constraint on a syntax element, wherein the syntax element corresponds to a chroma format of the video.

[00159] B24. The method of solution B23, wherein the constraint specifies that the syntax element is a non-negative integer less than a maximum value that equals a value of the syntax element plus one.

[00160] B25. The method of solution B24, wherein the syntax element being less than 2 imposes the constraint.

[00161] B26. The method of solution B24, wherein the syntax element being equal to 2 does not impose the constraint.

[00162] B27. The method of any of solutions B23 to B26, wherein the syntax element is `max_chroma_format_constraint_idc`.

[00163] B28. The method of any of solutions B1 to B27, wherein the conversion comprises decoding the video from the bitstream.

[00164] B29. The method of any of solutions B1 to B27, wherein the conversion comprises encoding the video into the bitstream.

[00165] B30. A method of storing a bitstream representing a video to a computer-readable recording medium, comprising generating the bitstream from the video according to a method described in any one or more of solutions B1 to B27; and storing the bitstream in the computer-readable recording medium.

[00166] B31. A video processing apparatus comprising a processor configured to implement a method recited in any one or more of solutions B1 to B30.

[00167] B32. A computer-readable medium having instructions stored thereon, the instructions, when executed, causing a processor to implement a method recited in one or more of solutions B1 to B30.

[00168] B33. A computer readable medium that stores the bitstream generated according to any one or more of solutions B1 to B30.

[00169] B34. A video processing apparatus for storing a bitstream, wherein the video processing apparatus is configured to implement a method recited in any one or more of solutions B1 to B30.

[00170] Yet another listing of solutions preferred by some embodiments is provided next.

[00171] P1. A video processing method, comprising performing a conversion between a video and a coded representation of the video, wherein the coded representation conforms to a format rule, wherein the format rule specifies where and how a general constraint information syntax (GCI) field is included in the coded representation, or a condition under which the GCI field is included in the coded representation.

[00172] P2. The method of solution P1, wherein, the rule specifies that the GCI field is included after an indication of a level information for the video.

[00173] P3. The method of any of solutions P1 to P2, wherein, the rule specifies to include the GCI field after a field indicating whether the GCI field is included in the coded representation.

[00174] P4. The method of solution P1, wherein the rule specifies that the GCI field is included after a tier indication flag and before a byte alignment syntax element in a profile-tier-level syntax structure.

[00175] P5. The method of any of solutions P1 to P4, wherein the rule further specifies that the coded representation includes a number of constraint bits or a value of each reserved constrain instead of signaling a total number of reserved constraint bytes.

[00176] P6. A video processing method, comprising performing a conversion between a video and a coded representation of the video, wherein the coded representation conforms to a format rule, wherein the format rule specifies a constraint on a syntax element, wherein the syntax element corresponds to a bit depth used for representing the video in the coded representation or a constrain of a chroma format of the video.

[00177] P7. The method of solution P6, wherein the format rule specifies a constraint for a value of the field that is less than 8.

[00178] P8. The method of solution P6, wherein the format rule specifies that the constraint is not imposed for a value 2 of the syntax element.

[00179] P9. The method of any of solutions P1 to P8, wherein the performing the conversion comprises encoding the video to generate the coded representation.

[00180] P10. The method of any of solutions P1 to P8, wherein the performing the conversion comprises parsing and decoding the coded representation to generate the video.

[00181] P11. A video decoding apparatus comprising a processor configured to implement a method recited in one or more of solutions P1 to P10.

[00182] P12. A video encoding apparatus comprising a processor configured to implement a method recited in one or more of solutions P1 to P10.

[00183] P13. A computer program product having computer code stored thereon, the code, when executed by a processor, causes the processor to implement a method recited in any of solutions P1 to P10.

[00184] In the present document, the term “video processing” may refer to video encoding, video decoding, video compression or video decompression. For example, video compression algorithms may be applied during conversion from pixel representation of a video to a corresponding bitstream representation or vice versa. The bitstream representation (or simply, the bitstream) of a current video block may, for example, correspond to bits that are either co-located or spread in different places within the bitstream, as is defined by the syntax. For example, a macroblock may be encoded in terms of transformed and coded error residual values and also using bits in headers and other fields in the bitstream.

[00185] The disclosed and other solutions, examples, embodiments, modules and the functional operations described in this document can be implemented in digital electronic circuitry, or in computer software, firmware, or hardware, including the structures disclosed in this document and their structural equivalents, or in combinations of one or more of them. The disclosed and other embodiments can be implemented as one or more computer program products, i.e., one or more modules of computer program instructions encoded on a computer readable medium for execution by, or to control the operation of, data processing apparatus. The computer readable medium can be a machine-readable storage device, a machine-readable storage substrate, a

memory device, a composition of matter effecting a machine-readable propagated signal, or a combination of one or more them. The term “data processing apparatus” encompasses all apparatus, devices, and machines for processing data, including by way of example a programmable processor, a computer, or multiple processors or computers. The apparatus can include, in addition to hardware, code that creates an execution environment for the computer program in question, e.g., code that constitutes processor firmware, a protocol stack, a database management system, an operating system, or a combination of one or more of them. A propagated signal is an artificially generated signal, e.g., a machine-generated electrical, optical, or electromagnetic signal, that is generated to encode information for transmission to suitable receiver apparatus.

[00186] A computer program (also known as a program, software, software application, script, or code) can be written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program does not necessarily correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data (e.g., one or more scripts stored in a markup language document), in a single file dedicated to the program in question, or in multiple coordinated files (e.g., files that store one or more modules, sub programs, or portions of code). A computer program can be deployed to be executed on one computer or on multiple computers that are located at one site or distributed across multiple sites and interconnected by a communication network.

[00187] The processes and logic flows described in this document can be performed by one or more programmable processors executing one or more computer programs to perform functions by operating on input data and generating output. The processes and logic flows can also be performed by, and apparatus can also be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit).

[00188] Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read only memory or a random-access memory or both. The essential elements of a computer are a processor for performing instructions and one or more memory devices for storing

instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto optical disks, or optical disks. However, a computer need not have such devices. Computer readable media suitable for storing computer program instructions and data include all forms of non-volatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto optical disks; and CD ROM and DVD-ROM disks. The processor and the memory can be supplemented by, or incorporated in, special purpose logic circuitry.

[00189] While this patent document contains many specifics, these should not be construed as limitations on the scope of any subject matter or of what may be claimed, but rather as descriptions of features that may be specific to particular embodiments of particular techniques. Certain features that are described in this patent document in the context of separate embodiments can also be implemented in combination in a single embodiment. Conversely, various features that are described in the context of a single embodiment can also be implemented in multiple embodiments separately or in any suitable subcombination. Moreover, although features may be described above as acting in certain combinations and even initially claimed as such, one or more features from a claimed combination can in some cases be excised from the combination, and the claimed combination may be directed to a subcombination or variation of a subcombination.

[00190] Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. Moreover, the separation of various system components in the embodiments described in this patent document should not be understood as requiring such separation in all embodiments.

[00191] Only a few implementations and examples are described and other implementations, enhancements and variations can be made based on what is described and illustrated in this patent document.