

[Fig. 1A]

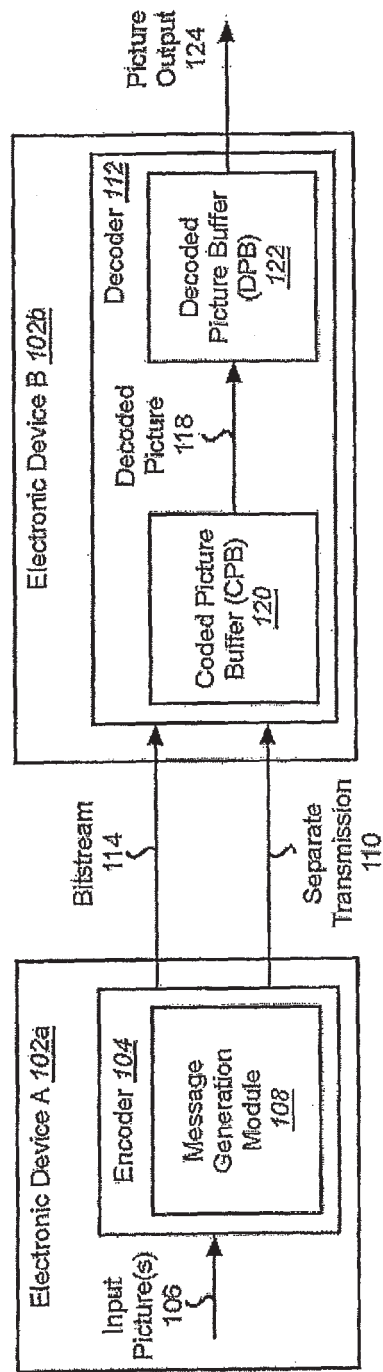


FIG. 1A

[Fig. 1B]

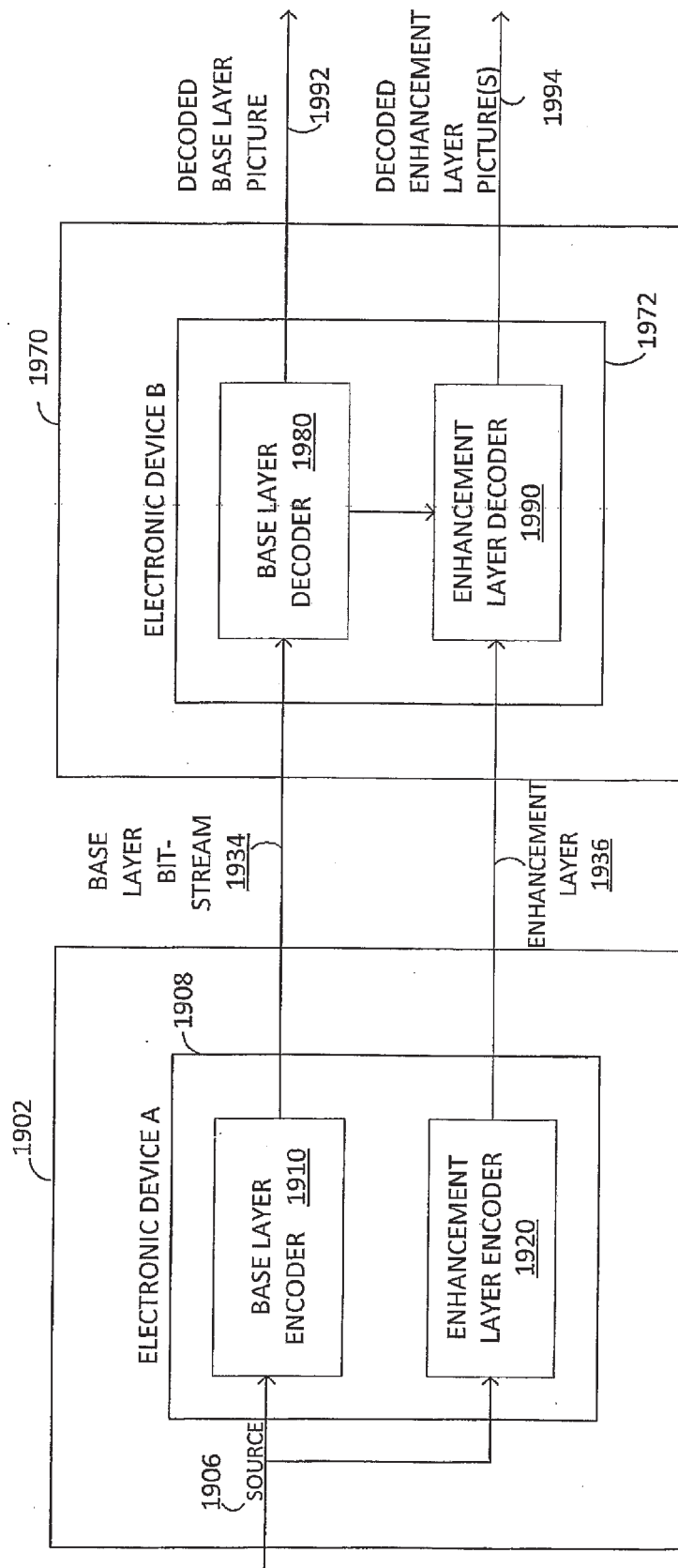


FIG. 1B

[Fig. 2]

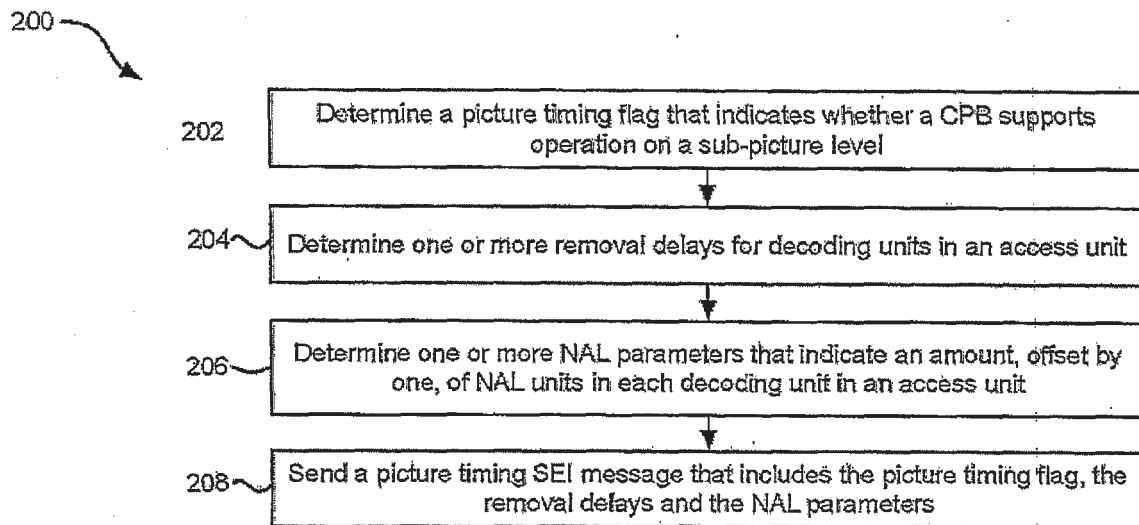


FIG. 2

[Fig. 3]

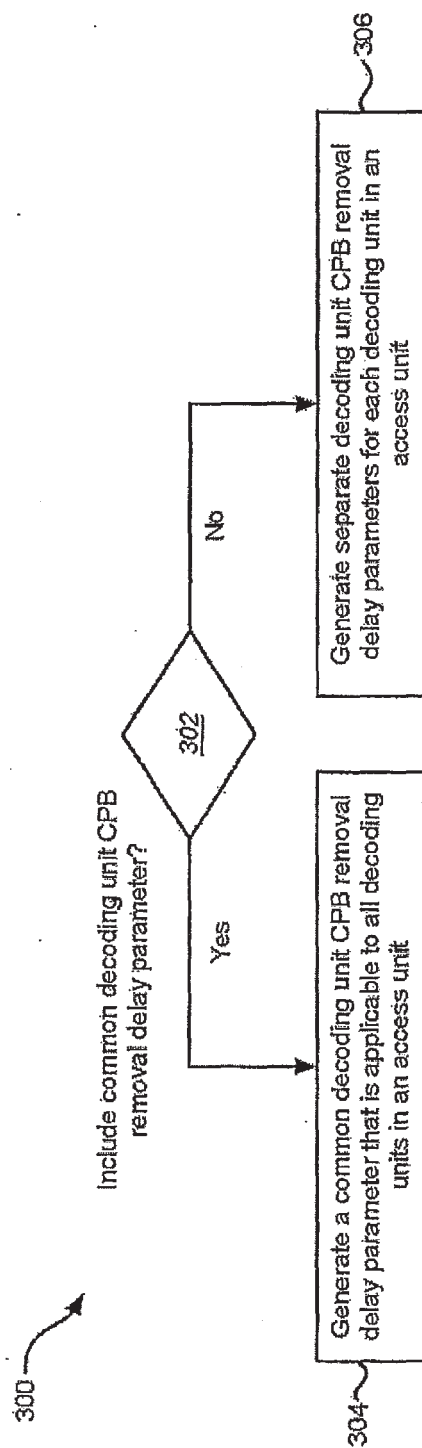


FIG. 3

[Fig. 4]

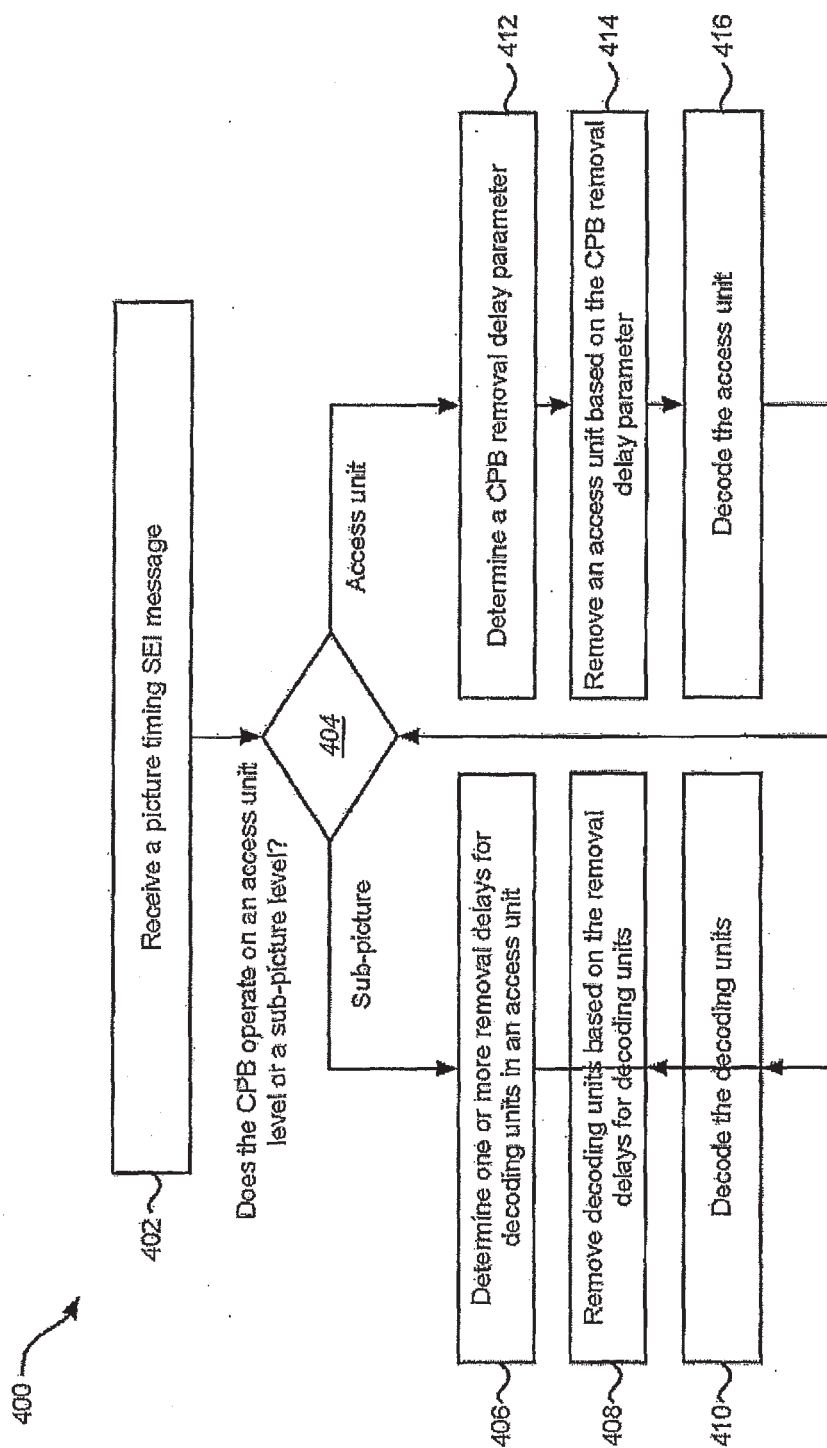


FIG. 4

[Fig. 5]

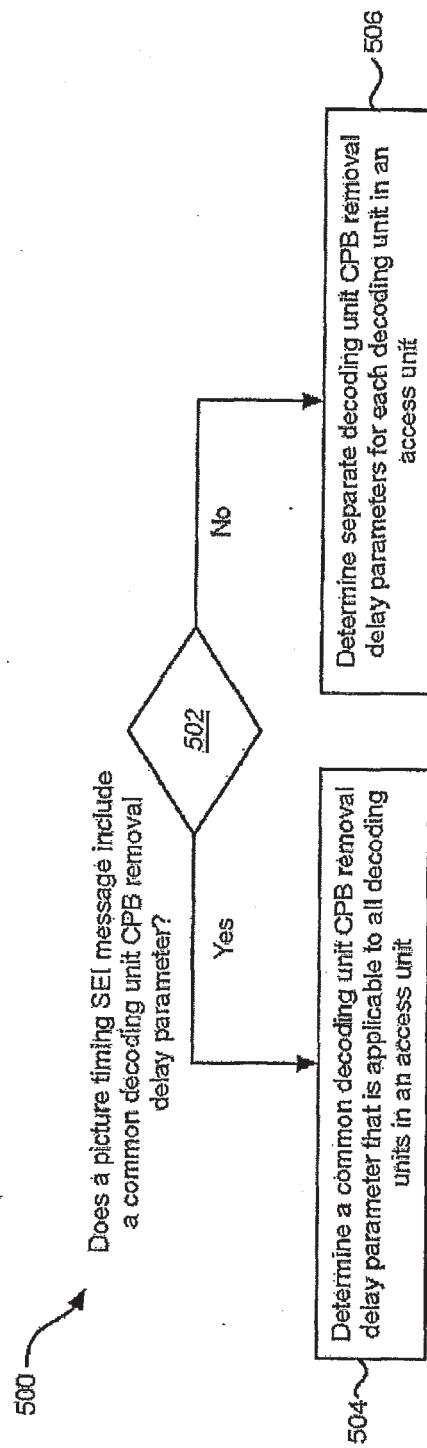


FIG. 5

[Fig. 6A]

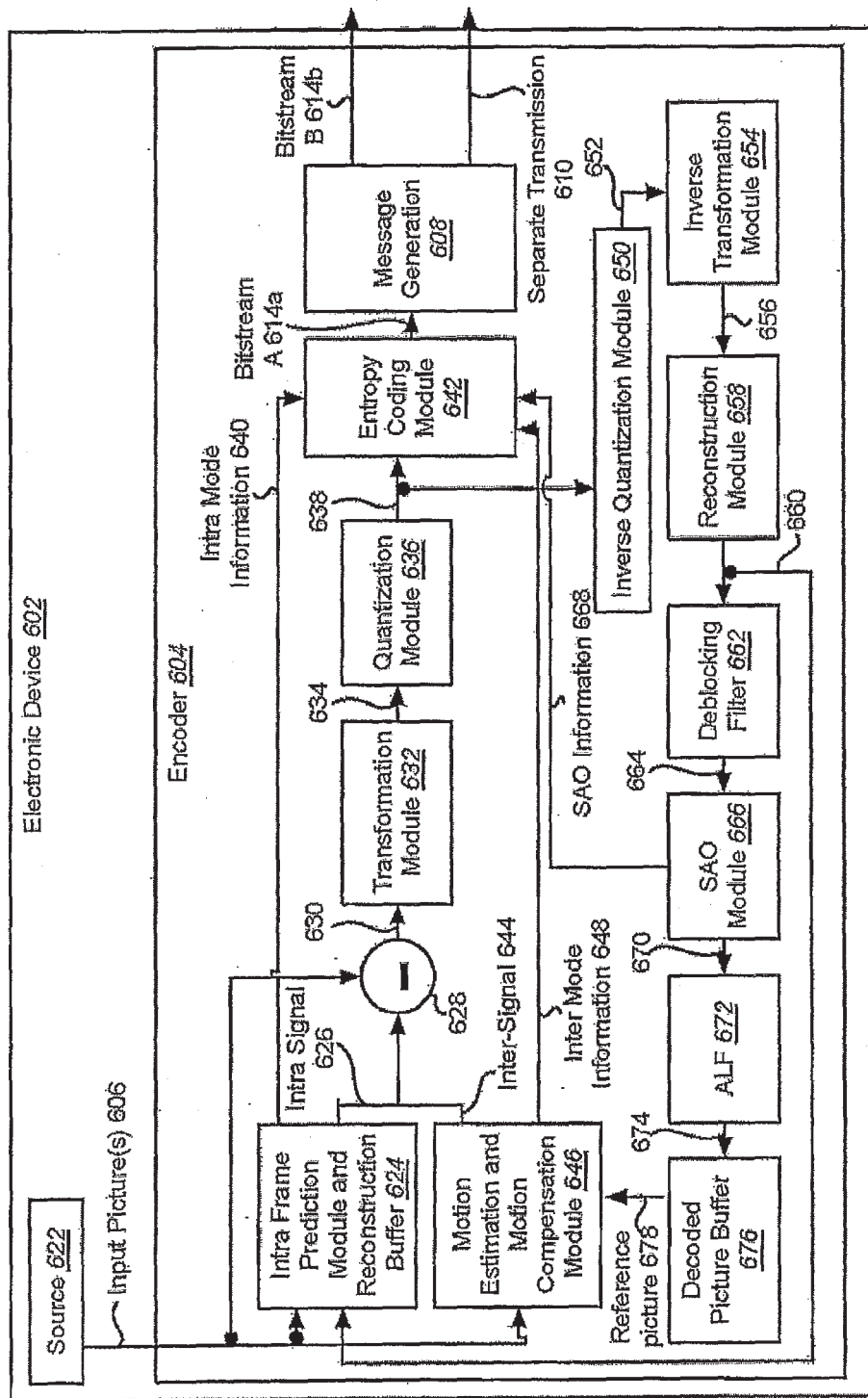
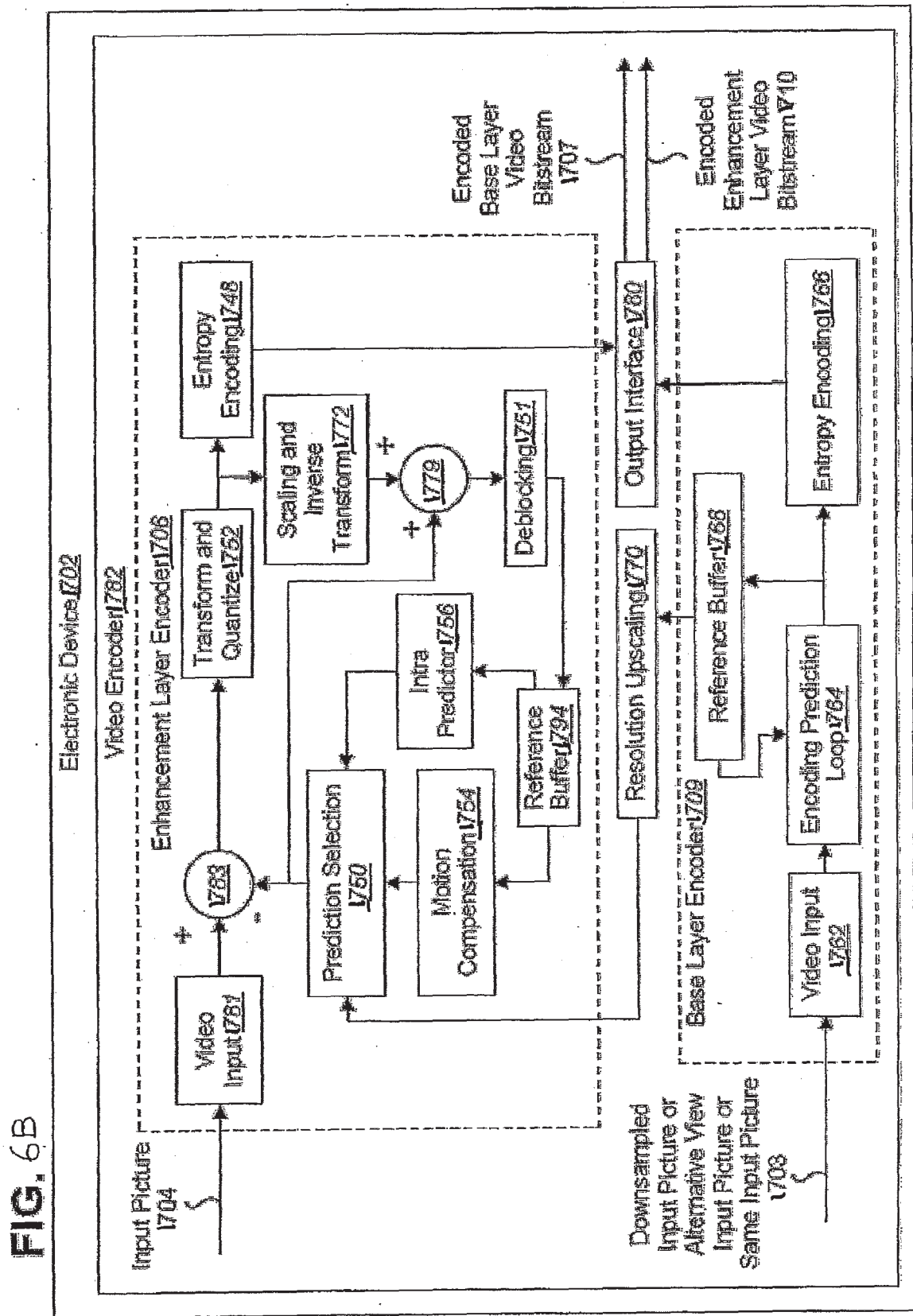


FIG. 6A

[Fig. 6B]



[Fig. 7A]

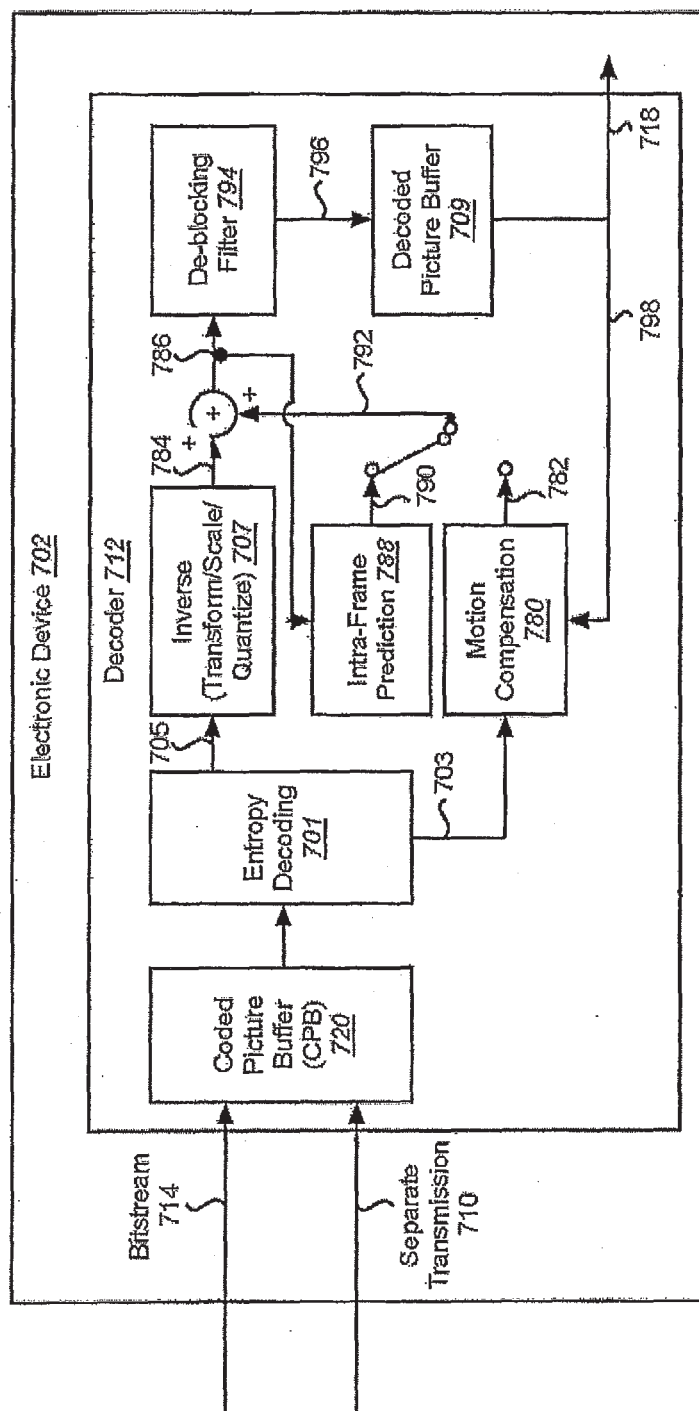
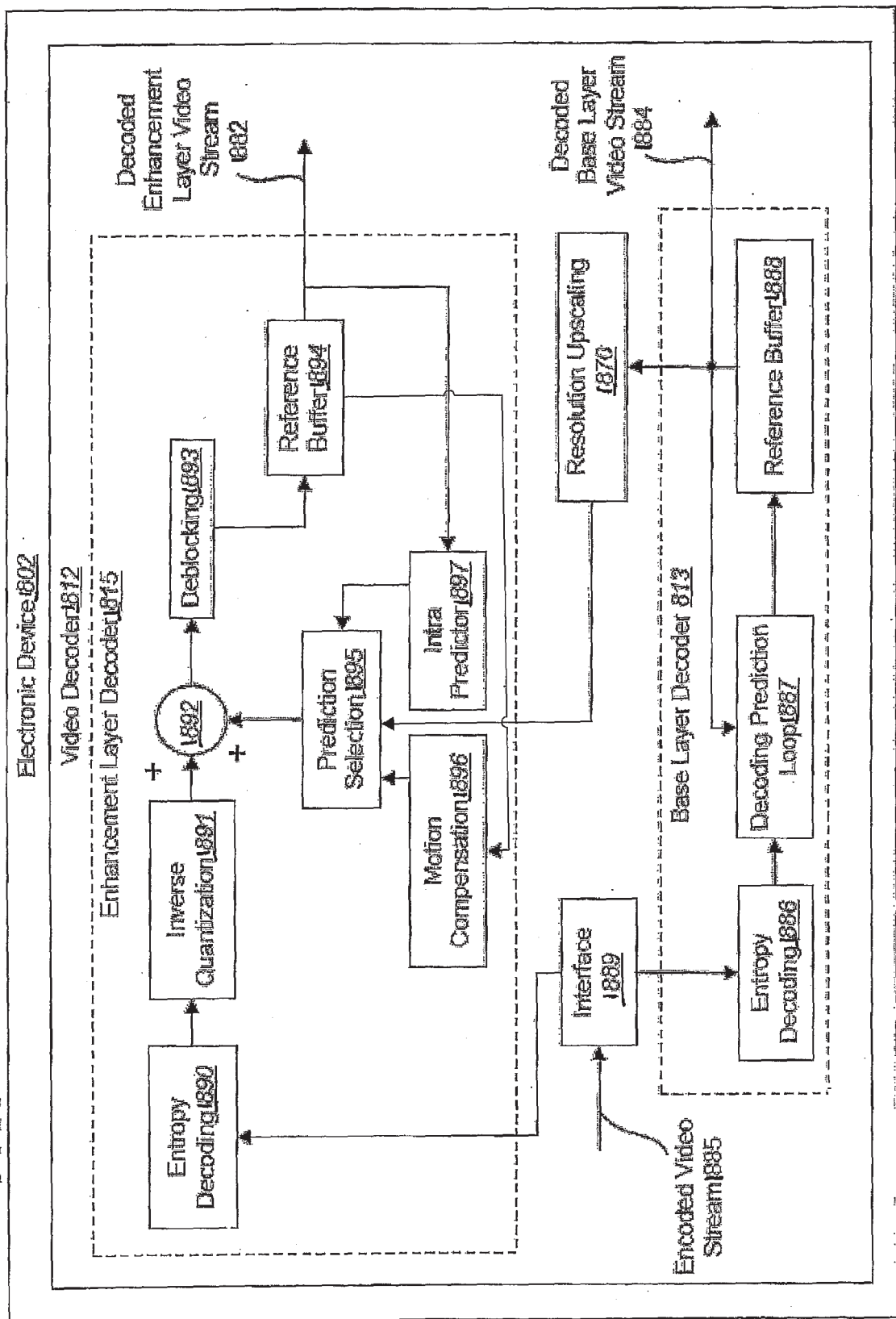


FIG. 7A

[Fig. 7B]

FIG. 7B



[Fig. 8]

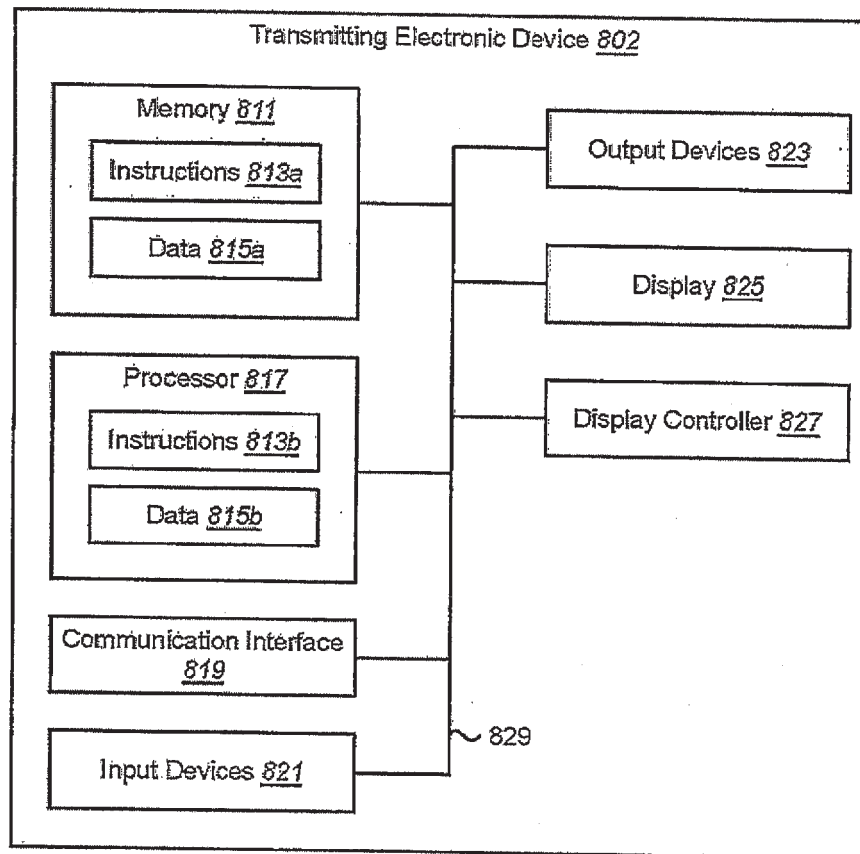


FIG. 8

[Fig. 9]

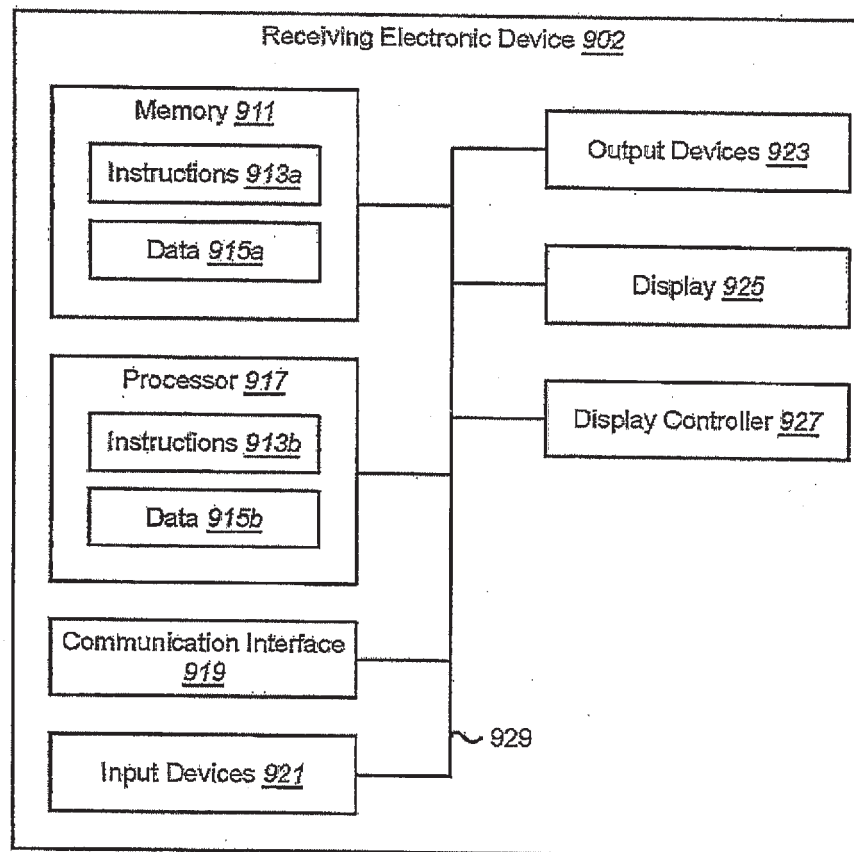


FIG. 9

[Fig. 10]

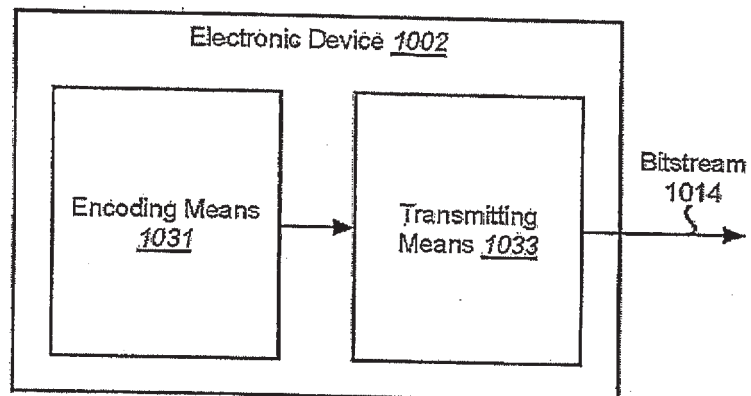


FIG. 10

[Fig. 11]

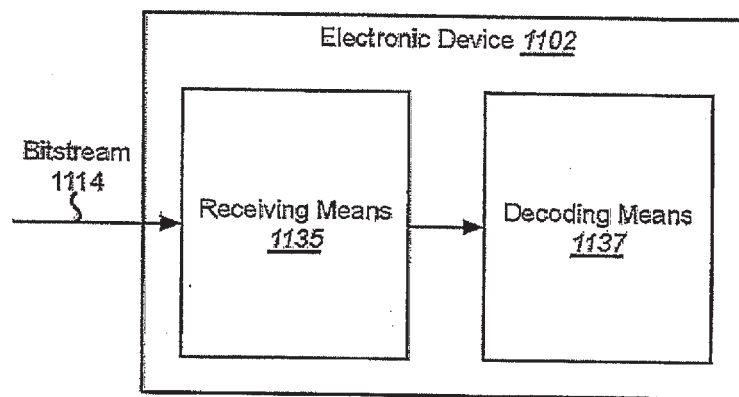


FIG. 11

[Fig. 12]

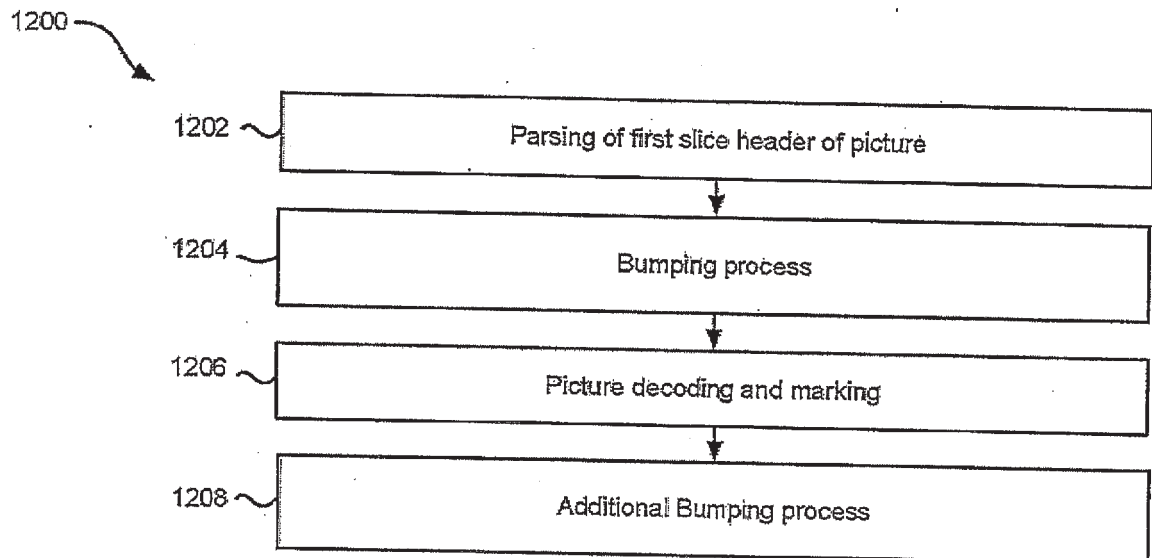


FIG. 12

[Fig. 13A]

nal_unit_header() {	
forbidden_zero_bit	f(1)
nal_unit_type	u(6)
nuh_reserved_zero_6bits	u(6)
nuh_temporal_id_plus1	u(3)
}	

NAL UNIT HEADER SYNTAX

FIG. 13A

[Fig. 13B]

nal_unit_header() {	Descriptor
forbidden_zero_bit	f(1)
nal_unit_type	u(6)
layer_id_plus1	u(6)
nuh_temporal_id_plus1	u(3)
}	

NAL UNIT HEADER SYNTAX

FIG. 13B

[Fig. 13C]

nal_unit_header() {	Descriptor
forbidden_zero_bit	f(1)
nal_unit_type	u(6)
layer_id	u(6)
nuh_temporal_id_plus1	u(3)
}	

NAL UNIT HEADER SYNTAX

FIG. 13C

[Fig. 14]

nal_unit(NumBytesInNALunit) {	Descriptor
nal_unit_header() .	
NumBytesInRBSP = 0	
for(i = 2; i < NumBytesInNALunit; i++) {	
if(i + 2 < NumBytesInNALunit && next_bits(24) == 0x000003) {	
rbsp_byte[NumBytesInRBSP++]	b(8)
rbsp_byte[NumBytesInRBSP++]	b(8)
i += 2	
emulation_prevention_three_byte /* equal to 0x03 */	f(8)
} else	
rbsp_byte[NumBytesInRBSP++]	b(8)
}	
}	

GENERAL NAL UNIT SYNTAX

FIG. 14

[Fig. 15]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extansion_byte_alignment_reserved_zero_bit	u(1)
// scalability type and layer_id partitioning method	
scalability_type	u(4)
for(i=0; i < MaxDim(scalability_type); i++)	
layer_id_dim_len[i]	u(3)
// layer specific information	
for(i=0; i <= max_num_layers_minus1; i++) {	
vps_layer_id[i]	u(6)
// layer dependency	
num_direct_ref_layers[i]	u(6)
for(j=0; j < num_direct_ref_layers[i]; j++)	
ref_layer_id[i][j]	u(6)
}	
}	

Existing video parameter set extension syntax

FIG. 15

[Fig. 16]

scalability type	MaxDim(scalability type)	Scalability dimensions
0	1	none (base HEVC)
1	2	spatial and quality
2	3	spatial, quality, unspecified
3	4	spatial, quality, unspecified, unspecified
4	2	multiview and depth
5	3	multiview, depth, unspecified
6	4	multiview, depth, unspecified, unspecified
7	4	multiview, spatial, quality and depth
8	5	multiview, spatial, quality, depth, unspecified
9	6	multiview, spatial, quality, depth, unspecified, unspecified
10...15	reserved	reserved

Existing Scalability Types

FIG. 16

[Fig. 17]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_zero_bit	u(1)
// Scalability map and layer_id_plus1 mapping method	
for (i=0; i<= max_num_layers_minus1_bits; i++) {	
scalability_map[i]	u(3)
}	
// layer specific information	
for(i = 0; i <= max_num_layers_minus1; i++) {	
vps_layer_id[i]	u(6)
// layer dependency	
num_direct_ref_layers[i]	u(6)
for(j = 0; j < num_direct_ref_layers[i]; j++)	
ref_layer_id[i][j]	u(6)
}	
}	

Video parameter set extension syntax

FIG. 17

[Fig. 18]

Scalability_map[i]	Scalability dimension
0	none (base HEVC)
1	spatial
2	quality
3	depth
4	multiview
5	unspecified
6, 7	reserved

Scalability Map Syntax

FIG. 18

[Fig. 19]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_zero_bit	u(1)
// scalability map and layer_id_plus1 mapping method	
num_scalability_dimensions_minus1	u(3)
for(i=0; i <= num_scalability_dimensions_minus1; i++) {	
scalability_map[i]	u(3)
num_bits_for_scalability_map[i]	u(3)
}	
// layer specific information	
for(i=0; i <= max_num_layers_minus1; i++) {	
vps_layer_id[i]	u(6)
// layer dependency	
num_direct_ref_layers[i]	u(6)
for(j=0; j < num_direct_ref_layers[i]; j++)	
ref_layer_id[i][j]	u(6)
}	
}	

Video parameter set extension syntax

FIG. 19

[Fig. 20]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_zero_bit	u(1)
// layer specific information	
for(i=1; i <= vps_max_layers_minus1; i++) {	
// mapping of layer ID to scalability dimension IDs	
num_dimensions_minus1[i]	u(4)
for(j=0; j <= num_dimensions_minus1; j++) {	
dimension_type[i][j]	u(4)
dimension_id[i][j]	u(8)
}	
// layer dependency	
num_direct_ref_layers[i]	u(6)
for(j=0; j < num_direct_ref_layers[i]; j++)	
ref_layer_id[i][j]	u(6)
}	
}	

Existing video parameter set extension syntax

FIG. 20

[Fig. 21]

dimension_type[i][j]	dimension_id[i][j]
0	view order idx
1	depth flag
2	dependency ID
3	quality ID
4..15	reserved

Existing Dimension Type, Dimension ID Syntax

FIG. 21

[Fig. 22]

vps_extension() {	Descriptor
while(lbyte_aligned())	
vps_extension_byte_alignment_reserved_zero_bit	u(1)
// layer specific information	
for(i = 1; i <= max_num_layers_minus1; i++) {	
// mask signalling scalability types that are present for this layer ID	
scalability_mask	u(8)
for(j = 0; j <= num_scalability_types[i]; j++) {	
scalability_id[j]	u(8)
}	
// layer dependency	
num_direct_ref_layers[i]	u(6)
for(j = 0; j < num_direct_ref_layers[i]; j++)	
ref_layer_id[i][j]	u(6)
}	
}	

Video parameter set extension syntax

FIG. 22

[Fig. 23]

scalability_mask_bit[k]	Scalability dimension
0	spatial
1	quality
2	depth
3	multiview
4-15	Reserved

Scalability Map Syntax

FIG. 23

[Fig. 24]

video_parameter_set_rbsp() {	Descriptor
video_parameter_set_id	u(4)
vps_temporal_id_nesting_flag	u(1)
reserved_zero_2bits	u(2)
max_num_layers_minus1 // reserved_zero_6bits	u(6)
vps_max_sub_layers_minus1	u(3)
profile_level(1, vps_max_sub_layers_minus1)	
next_essential_info_byte_offset // reserved_zero_12bits	u(12)
for(i = 0; i <= vps_max_sub_layers_minus1; i++) {	
vps_max_dec_pic_buffering[i]	ue(v)
vps_max_num_reorder_pics[i]	ue(v)
vps_max_latency_increase[i]	ue(v)
}	
num_hrd_parameters	ue(v)
for(i = 0; i < num_hrd_parameters; i++) {	
if(i > 0)	
op_point(i)	
hrd_parameters(i == 0, vps_max_sub_layers_minus1)	
}	
bit_equal_to_one	u(1)
scalability_mask	u(8)
for(i = 0; i <= max_num_layers_minus1; i++) {	
vps_extension_data()	
}	
vps_extension_flag	u(1)
if(vps_extension_flag)	
while(more_rbsp_data())	
vps_extension_data_flag	u(1)
rbsp_trailing_bits()	
}	
}	

Video parameter set extension syntax

FIG. 24

[Fig. 25]

vps_extension_data() {	Descriptor
vps_extension_type	u(2)
layer_id	u(6)
layer_max_sub_layers_minus1	u(3)
new_profile_level_flag	u(1)
if (new_profile_level_flag == 0) {	
profile_reference_layer_id	u(6)
}	
else {	
profile_level(1, layer_max_sub_layers_minus1)	
}	
for(j = 0; j <= num_scalability_types; j++) {	
scalable_id[j]	u(8)
}	
num_reference_layers	u(6)
for(n = 0; n < num_reference_layers; n++)	
direct_coding_dependency_layer_id_plus1[n]	u(6)
}	

Video parameter set extension syntax

FIG. 25

[Fig. 26]

	Descriptor
<code>vps_extension() {</code>	
<code>while(!byte_aligned())</code>	
<code>vps_extension_byte_alignment_reserved_one_bit</code>	<code>u(1)</code>
<code>ave_base_codec_flag</code>	<code>u(1)</code>
<code>scalability_mask</code>	<code>u(16)</code>
<code>for(i = 0; i < NumScalabilityTypes; i++) {</code>	
<code>dimension_id_in_minus1[i]</code>	<code>u(3)</code>
<code>}</code>	
<code>vps_nuh_layer_id_present_flag</code>	<code>u(1)</code>
<code>// layer specific information</code>	
<code>for(i = 1; i <= vps_max_layers_minus1; i++) {</code>	
<code>// mapping of layer ID to scalability dimension IDs</code>	
<code>if(vps_nuh_layer_id_present_flag)</code>	
<code>layer_id_in_nuh[i]</code>	<code>u(6)</code>
<code>for(j = 0; j < NumScalabilityTypes; j++)</code>	
<code>dimension_id[i][j]</code>	<code>u(v8)</code>
<code>}</code>	
<code>for(i = 1; i <= vps_max_layers_minus1; i++)</code>	
<code>profile_for_level(1, vps_max_sub_layers_minus1)</code>	
<code>for(i = 1; i <= vps_max_layers_minus1; i++) {</code>	
<code>// layer dependency</code>	
<code>num_direct_ref_layers[i]</code>	<code>u(6)</code>
<code>for(j = 0; j < num_direct_ref_layers[i]; j++)</code>	
<code>ref_layer_id[i][j]</code>	<code>u(6)</code>
<code>}</code>	
<code>}</code>	

Video Parameter Set Extension Syntax

FIG. 26

[Fig. 27]

scalability_mask's k'th bit	Scalability dimension
0	spatial
1	quality
2	depth
3	multiview
4-15	Reserved

Scalability_mask Syntax

FIG. 27

[Fig. 28]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
avc_base_codec_flag	u(1)
scalability_mask	u(16)
for(i = 0; i < NumScalabilityTypes; i++) {	
dimension_id_len_minus1[i]	u(3)
}	
vps_nuh_layer_id_present_flag	u(1)
// layer specific information	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
// mapping of layer ID to scalability dimension IDs	
if(vps_nuh_layer_id_present_flag)	
layer_id_in_nuh[i]	u(6)
for(j = 0; j <= num_dimensions_minus1; j++)	
dimension_id[i][j]	u(v)
}	
for(i = 1; i <= vps_max_layers_minus1; i++)	
profile_tier_level(1, vps_max_sub_layers_minus1)	
layer_dependency_information_pattern	u(v)
for(i = 1; i <= NumDepLayers; i++) {	
// layer dependency information signaling	
num_direct_ref_layers[i]	u(6)
for(j = 0; j < num_direct_ref_layers[i]; j++)	
ref_layer_id[i][j]	u(6)
}	
}	

Video Parameter Set Extension Syntax

FIG. 28

[Fig. 29]

	Descriptor
vps_extension() {	
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
avc_base_codec_flag	u(1)
scalability_mask	u(16)
for(i = 0; i < NumScalabilityTypes; i++) {	
dimension_id_len_minus1[i]	u(3)
}	
vps_nuh_layer_id_present_flag	u(1)
// layer specific information	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
// mapping of layer ID to scalability dimension IDs	
if(vps_nuh_layer_id_present_flag)	
layer_id_in_nuh[i]	u(6)
for(j = 0; j <= num_dimensions_minus1; j++)	
dimension_id[i][j]	u(v)
}	
for(i = 1; i <= vps_max_layers_minus1; i++)	
profile_tier_level(1, vps_max_sub_layers_minus1)	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
// layer dependency information signaling	
layer_dependency_flag[i]	u(1)
if(layer_dependency_flag[i]) {	
num_direct_ref_layers[i]	u(6)
for(j = 0; j < num_direct_ref_layers[i]; j++)	
ref_layer_id[i][j]	u(6)
}	
}	
}	
}	

Video Parameter Set Extension Syntax

FIG. 29

[Fig. 30]

	Descriptor
vps_extension() {	
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
avc_base_codec_flag	u(1)
scalability_mask	u(16)
for(i = 0; i < NumScalabilityTypes; i++) {	
dimension_id_len_minus1[i]	u(3)
}	
vps_nuh_layer_id_present_flag	u(1)
// layer specific information	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
// mapping of layer ID to scalability dimension IDs	
if(vps_nuh_layer_id_present_flag)	
layer_id_in_nuh[i]	u(6)
for(j = 0; j <= num_dimensions_minus1; j++)	
dimension_id[i][j]	u(v)
}	
for(i = 1; i <= vps_max_layers_minus1; i++)	
profile_tier_level(i, vps_max_sub_layers_minus1)	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
// layer dependency information signaling	
layer_dependency_map[i]	u(v)
}	
}	

Video Parameter Set Extension Syntax

FIG. 30

[Fig. 31]

	Descriptor
vps_extension() {	
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
avc_base_codec_flag	u(1)
scalability_mask	u(16)
for(i = 0; i < NumScalabilityTypes; i++) {	
dimension_id_len_minus1[i]	u(3)
}	
vps_nuh_layer_id_present_flag	u(1)
// layer specific information	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
// mapping of layer ID to scalability dimension IDs	
if(vps_nuh_layer_id_present_flag)	
layer_id_in_nuh[i]	u(6)
for(j = 0; j <= num_dimensions_minus1; j++)	
dimension_id[i][j]	u(v)
}	
for(i = 1; i <= vps_max_layers_minus1 ; i++)	
profile_tier_level(1, vps_max_sub_layers_minus1)	
layer_dependency_information_pattern	u(v)
for(i = 1; i <= NumDepLayers; i++)	
layer_dependency_map[i]	u(v)
}	

Video Parameter Set Extension Syntax

FIG. 31

[Fig. 32]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
avc_base_codec_flag	u(1)
scalability_mask	u(16)
for(i = 0; i < NumScalabilityTypes; i++) {	
dimension_id_len_minus1[i]	u(3)
}	
vps_nuh_layer_id_present_flag	u(1)
// layer specific information	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
// mapping of layer ID to scalability dimension IDs	
if(vps_nuh_layer_id_present_flag)	
layer_id_in_nuh[i]	u(6)
for(j = 0; j <= num_dimensions_minus1; j++)	
dimension_id[i][j]	u(v)
}	
for(i = 1; i <= vps_max_layers_minus1; i++)	
profile_tier_level(1, vps_max_sub_layers_minus1)	
layer_dependency_information_pattern	u(v)
for(i = 1; i <= vps_max_layers_minus1; i++) {	
if(layer_dependency_information_pattern(i)) {	
// layer dependency information signaling	
num_direct_ref_layers[i]	u(6)
for(j = 0; j < num_direct_ref_layers[i]; j++)	
ref_layer_id[i][j]	u(6)
}	
}	
}	

Video Parameter Set Extension Syntax

FIG. 32

[Fig. 33]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
avc_base_codec_flag	u(1)
scalability_mask	u(16)
for(i = 0; i < NumScalabilityTypes; i++) {	
dimension_id_len_minus1[i]	u(3)
}	
vps_nuh_layer_id_present_flag	u(1)
// layer specific information	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
// mapping of layer ID to scalability dimension IDs	
if(vps_nuh_layer_id_present_flag)	
layer_id_in_nuh[i]	u(6)
for(j = 0; j <= num_dimensions_minus1; j++)	
dimension_id[i][j]	u(v)
}	
for(l = 1; l <= vps_max_layers_minus1; l++)	
profile_tier_level(1, vps_max_sub_layers_minus1)	
layer_dependency_information_pattern	u(v)
for(i = 1; i <= vps_max_layers_minus1; i++) {	
if(layer_dependency_information_pattern(i)) {	
layer_dependency_map[i]	u(v)
}	
}	

Video Parameter Set Extension Syntax

FIG. 33

[Fig. 34]

video_parameter_set_rbsp() {	Descriptor
vps_video_parameter_set_id	u(4)
vps_reserved_three_2bits	u(2)
vps_max_layers_minus1	u(6)
vps_max_sub_layers_minus1	u(3)
vps_temporal_id_nesting_flag	u(1)
vps_extension_offset // vps_reserved_0xffff_16bits	u(16)
profile_tier_level(1, vps_max_sub_layers_minus1)	
vps_sub_layer_ordering_info_present_flag	u(1)
for(i = (vps_sub_layer_ordering_info_present_flag ? 0 : vps_max_sub_layers_minus1); i <= vps_max_sub_layers_minus1; i++) {	
vps_max_dec_pic_buffering_minus1[i]	ue(v)
vps_max_num_reorder_pics[i]	ue(v)
vps_max_latency_increase_plus1[i]	ue(v)
}	
vps_max_layer_id	u(6)
vps_num_layer_sets_minus1	ue(v)
for(i = 1; i <= vps_num_layer_sets_minus1; i++)	
for(j = 0; j <= vps_max_layer_id; j++)	
layer_id_included_flag[i][j]	u(1)
vps_timing_info_present_flag	u(1)
if(vps_timing_info_present_flag) {	
vps_num_units_in_tick	u(32)
vps_time_scale	u(32)
vps_poc_proportional_to_timing_flag	u(1)
if(vps_poc_proportional_to_timing_flag)	
vps_num_ticks_poc_diff_one_minus1	ue(v)
vps_num_hrd_parameters	ue(v)
for(i = 0; i < vps_num_hrd_parameters; i++) {	
hrd_layer_set_idx[i]	ue(v)
if(i > 0)	
cprms_present_flag[i]	u(1)
hrd_parameters(cprms_present_flag[i], vps_max_sub_layers_minus1)	
}	
}	
vps_extension_flag	u(1)
if(vps_extension_flag) {	
vps_extension()	
vps_extension2_flag	u(1)
if(vps_extension2_flag)	
while(more_rbsp_data())	
vps_extension_data_flag	u(1)
}	
rbsp_trailing_bits()	
}	

FIG. 34

[Fig. 35]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
avc_base_layer_flag	u(1)
splitting_flag	u(1)
for(i = 0, NumScalabilityTypes = 0; i < 16; i++) {	
scalability_mask[i]	u(1)
NumScalabilityTypes += scalability_mask[i]	
}	
for(j = 0; j < NumScalabilityTypes; j++)	
dimension_id_len_minus1[j]	u(3)
vps_nuh_layer_id_present_flag	u(1)
for(i = 1; i <= vps_max_layers_minus1; i++) {	
if(vps_nuh_layer_id_present_flag)	
layer_id_in_nuh[i]	u(6)
for(j = 0; j < NumScalabilityTypes; j++)	
dimension_id[i][j]	u(v)
}	
for(lsIdx = 1; lsIdx <= vps_num_layer_sets_minus1; lsIdx ++) {	
vps_profile_present_flag[lsIdx]	u(1)
if(!vps_profile_present_flag[lsIdx])	
profile_layer_set_ref_minus1[lsIdx]	ue(v)
profile_tier_level(vps_profile_present_flag[lsIdx], vps_max_sub_layers_minus1)	
}	
num_output_layer_sets	ue(v)
for(i = 0; i < num_output_layer_sets; i++) {	
output_layer_set_idx[i]	ue(v)
lsIdx = output_layer_set_idx[i]	
for(j = 0 ; j <= vps_max_layer_id; j++)	
if(layer_id_included_flag[lsIdx][j])	
output_layer_flag[lsIdx][j]	u(1)
}	
for(i = 1; i <= vps_max_layers_minus1; i++)	
for(j = 0; j < i; j++)	
direct_dependency_flag[i][j]	u(1)
}	

FIG. 35

[Fig. 36]

	Descriptor
output_layer_sets_change(payloadSize) {	
active_vps_id	u(4)
num_changed_output_layer_sets	ue(v)
for(i = 0; i < num_changed_output_layer_sets; i++) {	
changed_output_layer_set_idx_entry[i]	ue(v)
clsIdx = output_layer_set_idx[changed_output_layer_set_idx_entry[i]]	
for(j = 0; j <= vps_max_layer_id; j++)	
if(layer_id_included_flag[clsIdx][j])	
output_layer_flag[clsIdx][j]	u(1)
}	
num_addl_output_layer_sets	
for(i = num_output_layer_sets; i < num_output_layer_sets + num_addl_output_layer_sets; i++) {	
addlIdx = addl_output_layer_sets_idx[i]	ue(v)
for(j = 0; j <= vps_max_layer_id; j++)	
if(layer_id_included_flag[addlIdx][j])	
output_layer_flag[addlIdx][j]	u(1)
}	
}	

FIG. 36

[Fig. 37]

	Descriptor
output_layer_sets_change(payloadSize) {	
active_vps_id	u(4)
num_deleted_output_layer_sets	ue(v)
for(i = 0; i < num_deleted_output_layer_sets; i++)	
deleted_output_layer_sets_idx_entry[i]	ue(v)
num_changed_output_layer_sets	ue(v)
for(i = 0; i < num_changed_output_layer_sets; i++) {	
changed_output_layer_set_idx_entry[i]	ue(v)
clsIdx = output_layer_set_idx[changed_output_layer_set_idx_entry[i]]	
for(j = 0; j <= vps_max_layer_id; j++)	
if(layer_id_included_flag[clsIdx][j])	
output_layer_flag[clsIdx][j]	u(1)
}	
num_addl_output_layer_sets	
for(i = num_output_layer_sets - num_deleted_output_layer_sets; i < num_output_layer_sets - num_deleted_output_layer_sets + num_addl_output_layer_sets; i++) {	
addlIdx = addl_output_layer_sets_idx[i]	ue(v)
for(j = 0; j <= vps_max_layer_id; j++)	
if(layer_id_included_flag[addlIdx][j])	
output_layer_flag[addlIdx][j]	u(1)
}	
}	

FIG. 37

[Fig. 38A]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
avc_base_layer_flag	u(1)
splitting_flag	u(1)
for(i = 0, NumScalabilityTypes = 0; i < 16; i++) {	
scalability_mask[i]	u(1)
NumScalabilityTypes += scalability_mask[i]	
}	
for(j = 0; j < NumScalabilityTypes; j++)	
dimension_id_len_minus1[j]	u(3)
...	
num_output_layer_sets	ue(v)
for(i = 0; i < num_output_layer_sets; i++) {	
output_layer_set_idx[i]	ue(v)
lsIdx = output_layer_set_idx[i]	
for(j = 0; j <= vps_max_layer_id; j++)	
if(layer_id_included_flag[lsIdx][j])	
output_layer_flag[lsIdx][j]	u(1)
}	
num_op_dpb_info_parameters	ue(v)
for(i = 0; i < num_op_dpb_info_parameters; i++) {	
operation_point_layer_set_idx[i]	ue(v)
for(j = 0; j <= vps_max_layer_id; j++)	
op_dpb_info_parameters(j)	
}	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
for(j = 0; j < i; j++)	
direct_dependency_flag[i][j]	u(1)
}	

FIG. 38A

[Fig. 38B]

vps_extension() {	Descriptor
while(lbyte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
ave_base_layer_flag	u(1)
splitting_flag	u(1)
for(i = 0, NumScalabilityTypes = 0; i < 16; i++) {	
scalability_mask[i]	u(1)
NumScalabilityTypes += scalability_mask[i]	
}	
for(j = 0; j < NumScalabilityTypes; j++)	
dimension_id_len_minus1[j]	u(3)
...	
num_output_layer_sets	ue(v)
for(i = 0; i < num_output_layer_sets; i++) {	
output_layer_set_idx[i]	ue(v)
lsIdx = output_layer_set_idx[i]	
for(j = 0; j <= vps_max_layer_id; j++)	
if(layer_id_included_flag[lsIdx][j])	
output_layer_flag[lsIdx][j]	u(1)
}	
num_op_dpb_info_parameters	ue(v)
for(i = 0; i < num_op_dpb_info_parameters; i++) {	
operation_point_layer_set_idx[i]	ue(v)
for(j = 0; j <= vps_max_layer_id; j++)	
op_dpb_info_parameters(i, j)	
}	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
for(j = 0; j < i; j++)	
direct_dependency_flag[i][j]	u(1)
}	

FIG. 38B

[Fig. 39A]

op_dpb_info_parameters(j) {	
vps_max_sub_layers_minus1[j]	u(3)
vps_sub_layer_ordering_info_present_flag[j]	u(1)
for(k = (vps_sub_layer_ordering_info_present_flag[j] ? 0 : vps_max_sub_layers_minus1[j]);	
k <= vps_max_sub_layers_minus1[j]; k++) {	
vps_max_dec_pic_buffering_minus1[j][k]	ue(v)
vps_max_num_reorder_pics[j][k]	ue(v)
vps_max_latency_increase_plus1[j][k]	ue(v)
}	
}	

FIG. 39A

[Fig. 39B]

op_dpb_info_parameters(id, j) {	
vps_max_sub_layers_minus1[id][j]	u(3)
vps_sub_layer_ordering_info_present_flag[id][j]	u(1)
for(k = (vps_sub_layer_ordering_info_present_flag[id][j] ? 0 : vps_max_sub_layers_minus1[id][j]); k <= vps_max_sub_layers_minus1[id][j]; k++) {	
vps_max_dec_pic_buffering_minus1[id][j][k]	ue(v)
vps_max_num_reorder_pics[id][j][k]	ue(v)
vps_max_latency_increase_plus1[id][j][k]	ue(v)
}	
}	

FIG. 39B

[Fig. 40]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
avc_base_layer_flag	u(1)
splitting_flag	u(1)
for(i = 0, NumScalabilityTypes = 0; i < 16; i++) {	
scalability_mask[i]	u(1)
NumScalabilityTypes += scalability_mask[i]	
}	
for(j = 0; j < NumScalabilityTypes; j++)	
dimension_id_len_minus1[j]	u(3)
...	
num_output_layer_sets	ue(v)
for(i = 0; i < num_output_layer_sets; i++) {	
output_layer_set_idx[i]	ue(v)
lsIdx = output_layer_set_idx[i]	
for(j = 0; j <= vps_max_layer_id; j++)	
if(layer_id_included_flag[lsIdx][j])	
output_layer_flag[lsIdx][j]	u(1)
}	
num_dpb_info_parameters	ue(v)
for(i = 0; i < num_dpb_info_parameters; i++) {	
output_point_layer_set_idx[i]	ue(v)
oplsIdx = output_point_layer_set_idx[i]	
for(j = 0; j <= vps_max_layer_id; j++)	
oop_dpb_info_parameters(j)	
}	
for(i = 1; i <= vps_max_layers_minus1; i++)	
for(j = 0; j < i; j++)	
direct_dependency_flag[i][j]	u(1)
}	

FIG. 40

[Fig. 41]

oop_dpb_info_parameters(j) {	
vps_max_sub_layers_minus1[j]	u(3)
vps_sub_layer_ordering_info_present_flag[j]	u(1)
for(k = (vps_sub_layer_ordering_info_present_flag[j] ? 0 : vps_max_sub_layers_minus1[j]); k <= vps_max_sub_layers_minus1[j]; k++) {	
vps_max_dec_pic_buffering_minus1[j][k]	ue(v)
vps_max_num_reorder_pics[j][k]	ue(v)
vps_max_latency_increase_plus1[j][k]	ue(v)
}	
}	

FIG. 41

[Fig. 42]

oop_dpb_info_parameters() {	
vps_max_sub_layers_minus1	u(3)
vps_sub_layer_ordering_info_present_flag	u(1)
for(k = (vps_sub_layer_ordering_info_present_flag ? 0 : vps_max_sub_layers_minus1); k <= vps_max_sub_layers_minus1; k++) {	
vps_max_dec_pic_buffering_minus1 [k]	ue(v)
vps_max_num_reorder_pics [k]	ue(v)
vps_max_latency_increase_plus1 [k]	ue(v)
}	
}	

FIG. 42

[Fig. 43]

num_dpb_info_parameters	ue(v)
for(i = 0; i < num_dpb_info_parameters; i++) {	
output_point_layer_set_idx[i]	ue(v)
oplsIdx=output_point_layer_set_idx[i]	
for(j = 0; j <= vps_max_layer_id; j++)	
oop_dpb_info_parameters()	
}	

FIG. 43

[Fig. 44]

oop_dpb_info_parameters(id,j) {	
vps_max_sub_layers_minus1[id][j]	u(3)
vps_sub_layer_ordering_info_present_flag[id][j]	u(1)
for(k=(vps_sub_layer_ordering_info_present_flag[id][j]? 0: vps_max_sub_layers_minus1[id][j]); k <= vps_max_sub_layers_minus1[id][j]; k++) {	
vps_max_dec_pic_buffering_minus1[id][j][k]	ue(v)
vps_max_num_reorder_pics[id][j][k]	ue(v)
vps_max_latency_increase_plus1[id][j][k]	ue(v)
}	
}	

FIG. 44

[Fig. 45]

num_dpb_info_parameters	ue(v)
for(i=0; i<num_dpb_info_parameters; i++) {	
output_point_layer_set_idx[i]	ue(v)
oplsIdx=output_point_layer_set_idx[i]	
for(j=0; j<=vps_max_layer_id; j++)	
oop_dpb_info_parameters(oplsIdx,j)	
}	

FIG. 45

[Fig. 46]

num_dpb_info_parameters	ue(v)
for(i=0; i<num_dpb_info_parameters; i++) {	
output_point_layer_set_idx[i]	ue(v)
oplsIdx=output_point_layer_set_idx[i]	
for(j=0; j<=vps_max_layer_id; j++)	
oop_dpb_info_parameters(i,j)	
}	

FIG. 46

[Fig. 47]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
avc_base_layer_flag	u(1)
splitting_flag	u(1)
for(i = 0, NumScalabilityTypes = 0; i < 16; i++) {	
scalability_mask[i]	u(1)
NumScalabilityTypes += scalability_mask[i]	
}	
for(j = 0; j < NumScalabilityTypes; j++)	
dimension_id_len_minus1[j]	u(3)
...	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
for(j = 0; j < i; j++)	
direct_dependency_flag[i][j]	u(1)
for(i = 1; i <= vps_max_layers_minus1; i++) {	
layer_dpb_info(i)	
}	
}	

layer_dpb_info(i) {	
vps_max_sub_layers_minus1[i]	u(3)
vps_sub_layer_ordering_info_present_flag[i]	u(1)
for(k = (vps_sub_layer_ordering_info_present_flag[i] ? 0 : vps_max_sub_layers_minus1[i]); k <= vps_max_sub_layers_minus1[i]; k++) {	
vps_max_dec_pic_buffering_minus1[i][k]	ue(v)
vps_max_num_reorder_pics[i][k]	ue(v)
vps_max_latency_increase_plus1[i][k]	ue(v)
}	
}	

FIG. 47

[Fig. 48]

oop_dpb_info_parameters(id) / op_dpb_info_parameters(id) / layer_dpb_info(id) {	
vps_sub_layer_ordering_info_present_flag[id]	u(1)
for(i = (vps_sub_layer_ordering_info_present_flag ? 0 : vps_max_sub_layers_minus1); i <= vps_max_sub_layers_minus1; i++) {	
vps_max_dec_pic_buffering_minus1[id][i]	ue(v)
vps_max_num_reorder_pics[id][i]	ue(v)
vps_max_latency_increase_plus1[id][i]	ue(v)
}	
}	

FIG. 48

[Fig. 49A]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
avc_base_layer_flag	u(1)
splitting_flag	u(1)
for(i = 0, NumScalabilityTypes = 0; i < 16; i++) {	
scalability_mask[i]	u(1)
NumScalabilityTypes += scalability_mask[i]	
}	
for(j = 0; j < NumScalabilityTypes; j++)	
dimension_id_len_minus1[j]	u(3)
vps_nuh_layer_id_present_flag	u(1)
for(i = 1; i <= vps_max_layers_minus1; i++) {	
if(vps_nuh_layer_id_present_flag)	
layer_id_in_nuh[i]	u(6)
for(j = 0; j < NumScalabilityTypes; j++)	
dimension_id[i][j]	u(v)
}	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
max_sub_layers_vps_predict_flag[i]	u(1)
if(!max_sub_layers_vps_predict_flag[i])	
max_sub_layers_vps_minus1[i]	u(3)
}	
vps_number_layer_sets_minus1	u(10)
vps_num_profile_tier_level_minus1	u(6)
for(i = 1; i <= vps_num_profile_tier_level_minus1; i++) {	
vps_profile_present_flag[i]	u(1)
if(!vps_profile_present_flag[i])	
profile_ref_minus1[i]	u(6)
profile_tier_level(vps_profile_present_flag[i], vps_max_sub_layers_minus1)	
}	

FIG. 49A

[Fig. 49B]

multiple_output_layer_sets_in_layer_setmore_output_layer_sets_than_default_flag	u(1)
if(!multiple_output_layer_sets_in_layer_setmore_output_layer_sets_than_default_flag)	
numOutputLayerSets = vps_number_layer_sets_minus1 + 1	
else {	
num_add_output_layer_sets	u(10)
numOutputLayerSets = num_add_output_layer_sets + vps_number_layer_sets_minus1 + 1	
}	
if(numOutputLayerSets > 1)	
default_one_target_output_layer_flag	u(1)
for(i = 1; i < numOutputLayerSets; i++) {	
if(i > vps_num_layer_sets_minus1) {	
output_layer_set_idx_minus1[i]	u(v)
lsIdx = output_layer_set_idx_minus1[i] + 1	
for(j = 0 ; j < NumLayersInIdList[lsIdx] - 1; j++)	
output_layer_flag[i][j]	u(1)
}	
profile_level_tier_idx[i]	u(v)
}	
num_dpb_info_parameters	ue(v)
for(i = 0; i < num_dpb_info_parameters; i++) {	
output_point_layer_set_idx[i]	ue(v)
oop_dpb_maxbuffering_parameters(i)	
}	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
layer_dpb_info_parameters(i)	
}	
}	
for(i = 1; i <= vps_max_layers_minus1; i++)	
for(j = 0; j < i; j++)	
direct_dependency_flag[i][j]	u(1)
}	

FIG. 49B

[Fig. 50]

oop_dpb_maxbuffering_parameters (i) {	
sub_layer_vps_buf_info_present_flag[i]	u(1)
for(j = (sub_layer_vps_buf_info_present_flag[i] ? 0 : MaxSubLayers[i] - 1); j <= MaxSubLayers[i] - 1; j++)	
max_vps_dec_pic_buffering_minus1[i][j]	ue(v)
}	

FIG. 50

[Fig. 51]

layer_dpb_info_parameters(i) {	
sub_layer_vps_ordering_info_present_flag[i]	u(1)
for(j = (sub_layer_vps_ordering_info_present_flag[i] ? 0 : max_sub_layers_vps_minus1[i]); j <= max_sub_layers_vps_minus1[i]; j++) {	
max_vps_num_reorder_pics[i][j]	ue(v)
max_vps_latency_increase_plus1[i][j]	ue(v)
}	
}	

FIG. 51

[Fig. 52]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
...	
for(j = 0; j < NumScalabilityTypes; j++)	
dimension_id[i][j]	u(v)
}	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
max_sub_layers_vps_predict_flag[i]	u(1)
if(!max_sub_layers_vps_predict_flag[i])	
max_sub_layers_vps_minus1[i]	u(3)
}	
...	
profile_level_tier_idx[i]	u(v)
}	
num_dpb_info_parameters	ue(v)
for(i = 0; i < num_dpb_info_parameters; i++) {	
if(i>0 && (MaxSubLayers[i] == MaxSubLayers[i-1]))	
sub_layer_vps_buf_info_predict_flag[i]	u(1)
if(!sub_layer_vps_buf_info_predict_flag[i]) {	
output_point_layer_set_idx[i]	ue(v)
oop_dpb_maxbuffering_parameters(i)	
}	
}	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
if(max_sub_layers_vps_predict_flag[i])	
sub_layer_vps_ordering_info_predict_flag[i]	u(1)
if(!sub_layer_ordering_info_predict_flag[i])	
layer_dpb_info_parameters(i)	
}	
}	
...	
}	

FIG. 52

[Fig. 53]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
...	
for(j = 0; j < NumScalabilityTypes; j++)	
dimension_id[i][j]	u(v)
}	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
max_sub_layers_vps_predict_flag[i]	u(1)
if(!max_sub_layers_vps_predict_flag[i])	
max_sub_layers_vps_minus1[i]	u(3)
}	
...	
profile_level_tier_idx[i]	u(v)
}	
num_dpb_info_parameters	ue(v)
for(i = 0; i < num_dpb_info_parameters; i++) {	
output_point_layer_set_idx[i]	ue(v)
for(k = 1; k <= vps_max_layers_minus1; i++) {	
oop_dpb_maxbuffering_parameters(i, k)	
}	
}	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
layer_dpb_info_parameters(i)	
}	
}	
...	
}	

FIG. 53

[Fig. 54]

oop_dpb_maxbuffering_parameters(i, k) {	
sub_layer_vps_buf_info_present_flag[i][k]	u(1)
for(j = (sub_layer_vps_buf_info_present_flag[i][k] ? 0 : max_sub_layers_vps_minus1[k]); j <= max_sub_layers_vps_minus1[k]; j++); j++)	
max_vps_dec_pic_buffering_minus1[i][k][j]	ue(v)
}	

FIG. 54

[Fig. 55]

oop_dpb_maxbuffering_parameters(i, k) {	
sub_layer_vps_buf_info_present_flag[i][k]	u(1)
for(j = (sub_layer_vps_buf_info_present_flag[i] ? 0 : MaxSubLayers[i][k] - 1); j <= MaxSubLayers[i][k] - 1; j++)	
max_vps_dec_pic_buffering_minus1[i][k][j]	ue(v)
}	

FIG. 55

[Fig. 56]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
...	
for(j = 0; j < NumScalabilityTypes; j++)	
dimension_id[i][j]	u(v)
}	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
max_sub_layers_vps_predict_flag[i]	u(1)
if(!max_sub_layers_vps_predict_flag[i])	
max_sub_layers_vps_minus1[i]	u(3)
}	
...	
profile_level_tier_idx[i]	u(v)
}	
num_dpb_info_parameters	ue(v)
for(i = 0; i < num_dpb_info_parameters; i++) {	
if(i>0 && (MaxSubLayers[i] == MaxSubLayers[i - 1]))	
sub_layer_vps_buf_info_predict_flag[i]	u(1)
if(!sub_layer_vps_buf_info_predict_flag[i]) {	
output_point_layer_set_idx[i]	ue(v)
for(k = 1; k <= vps_max_layers_minus1; i++) {	
oop_dpb_maxbuffering_parameters(i, k)	
}	
}	
}	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
if(max_sub_layers_vps_predict_flag[i])	
sub_layer_vps_ordering_info_predict_flag[i]	u(1)
if(!sub_layer_ordering_info_predict_flag[i])	
layer_dpb_info_parameters(i)	
}	
}	
...	
}	

FIG. 56

[Fig. 57]

oop_dpb_maxbuffering_parameters (i, k) {	
sub_layer_vps_buf_info_present_flag[i][k]	u(1)
for(j = (sub_layer_vps_buf_info_present_flag[i][k] ? 0 : max_sub_layers_vps_minus1[k]); j <= max_sub_layers_vps_minus1[k]; j++); j++)	
max_vps_dec_pic_buffering_minus1[i][k][j]	ue(v)
}	

FIG. 57

[Fig. 58]

oop_dpb_maxbuffering_parameters (i, k) {	
sub_layer_vps_buf_info_present_flag[i][k]	u(1)
for(j = (sub_layer_vps_buf_info_present_flag[i] ? 0 : MaxSubLayers[i][k] - 1); j <= MaxSubLayers[i][k] - 1; j++)	
max_vps_dec_pic_buffering_minus1[i][k][j]	ue(v)
}	

FIG. 58

[Fig. 59]

vps_extension() {	Descriptor
while(!byte_aligned())	
vps_extension_byte_alignment_reserved_one_bit	u(1)
...	
for(j = 0; j < NumScalabilityTypes; j++)	
dimension_id[i][j]	u(v)
}	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
max_sub_layers_vps_predict_flag[i]	u(1)
if(!max_sub_layers_vps_predict_flag[i])	
max_sub_layers_vps_minus1[i]	u(3)
}	
...	
profile_level_tier_idx[i]	u(v)
}	
num_dpb_info_parameters	ue(v)
for(i = 0; i < num_dpb_info_parameters; i++) {	
if(i>0 && (MaxSubLayers[i] == MaxSubLayers[i-1]))	
sub_layer_vps_buf_info_predict_flag[i]	u(1)
if(!sub_layer_vps_buf_info_predict_flag[i]) {	
output_point_layer_set_idx[i]	ue(v)
for(k = 1; k <= vps_max_layers_minus1; i++) {	
oop_dpb_maxbuffering_parameters(i, k)	
}	
}	
}	
layer_dpb_info_parameters_presence_flag	u(1)
If(layer_dpb_info_parameters_presence_flag) {	
for(i = 1; i <= vps_max_layers_minus1; i++) {	
if(max_sub_layers_vps_predict_flag[i])	
sub_layer_vps_ordering_info_predict_flag[i]	u(1)
if(!sub_layer_ordering_info_predict_flag[i])	
layer_dph_info_parameters(i)	
}	
}	
}	
...	
}	

FIG. 59

[Fig. 60]

oop_dpb_maxbuffering_parameters (i, k) {	
sub_layer_vps_maxbuffering_info_present_flag[i][k]	u(1)
for(j = (sub_layer_vps_maxbuffering_info_present_flag[i][k] ? 0 : max_sub_layers_vps_minus1[k]); j <= max_sub_layers_vps_minus1[k]; j++)	
max_vps_dec_pic_buffering_minus1[i][k][j]	ue(v)
}	

FIG. 60

[Fig. 61]

oop_dpb_maxbuffering_parameters (i, k) {	
sub_layer_vps_buf_info_present_flag[i][k]	u(1)
for(j = (sub_layer_vps_buf_info_present_flag[i] ? 0 : MaxSubLayers[i][k] - 1); j <= MaxSubLayers[i][k] - 1; j++)	
max_vps_dec_pic_buffering_minus1[i][k][j]	ue(v)
}	

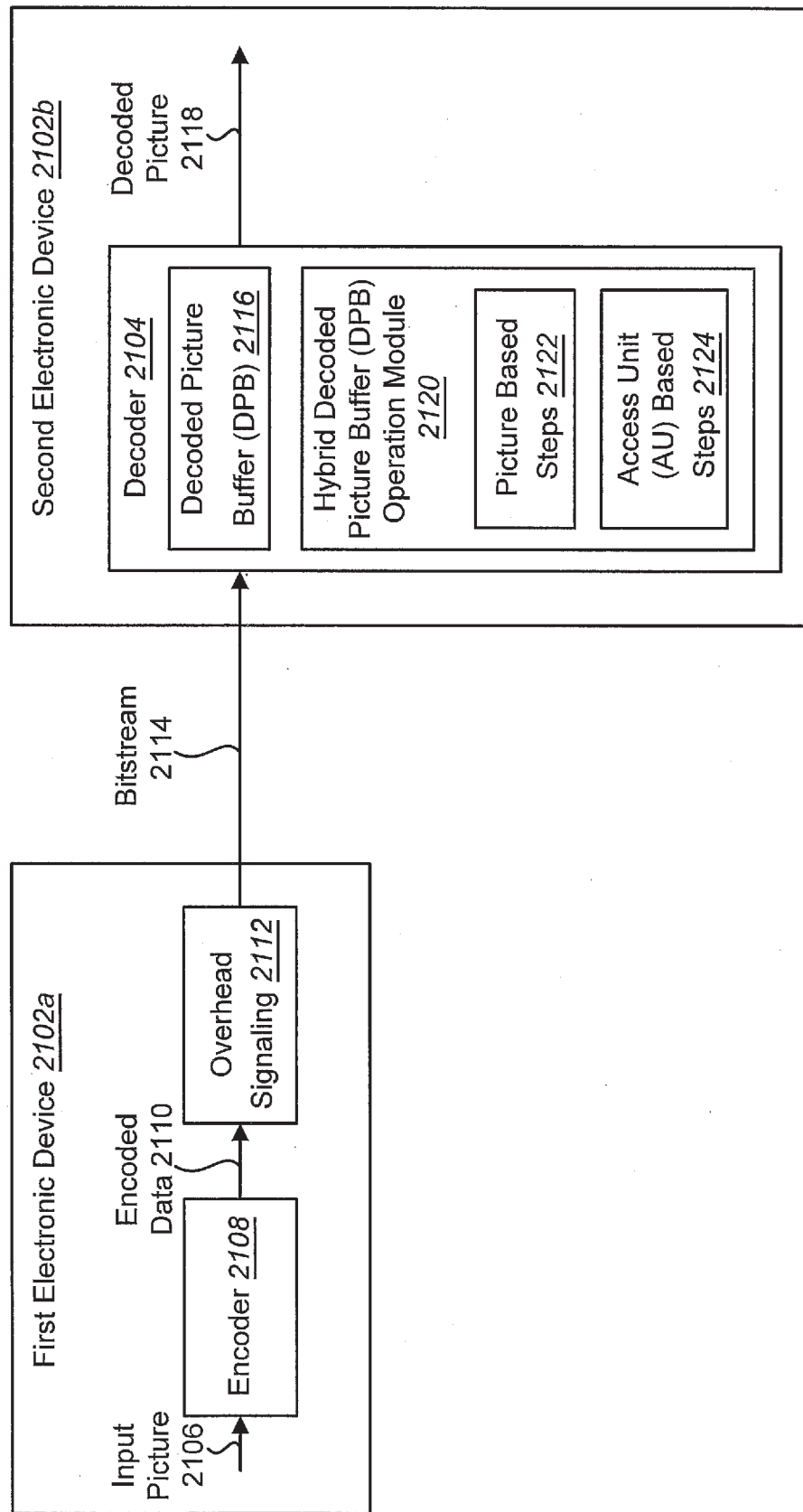
FIG. 61

[Fig. 62]

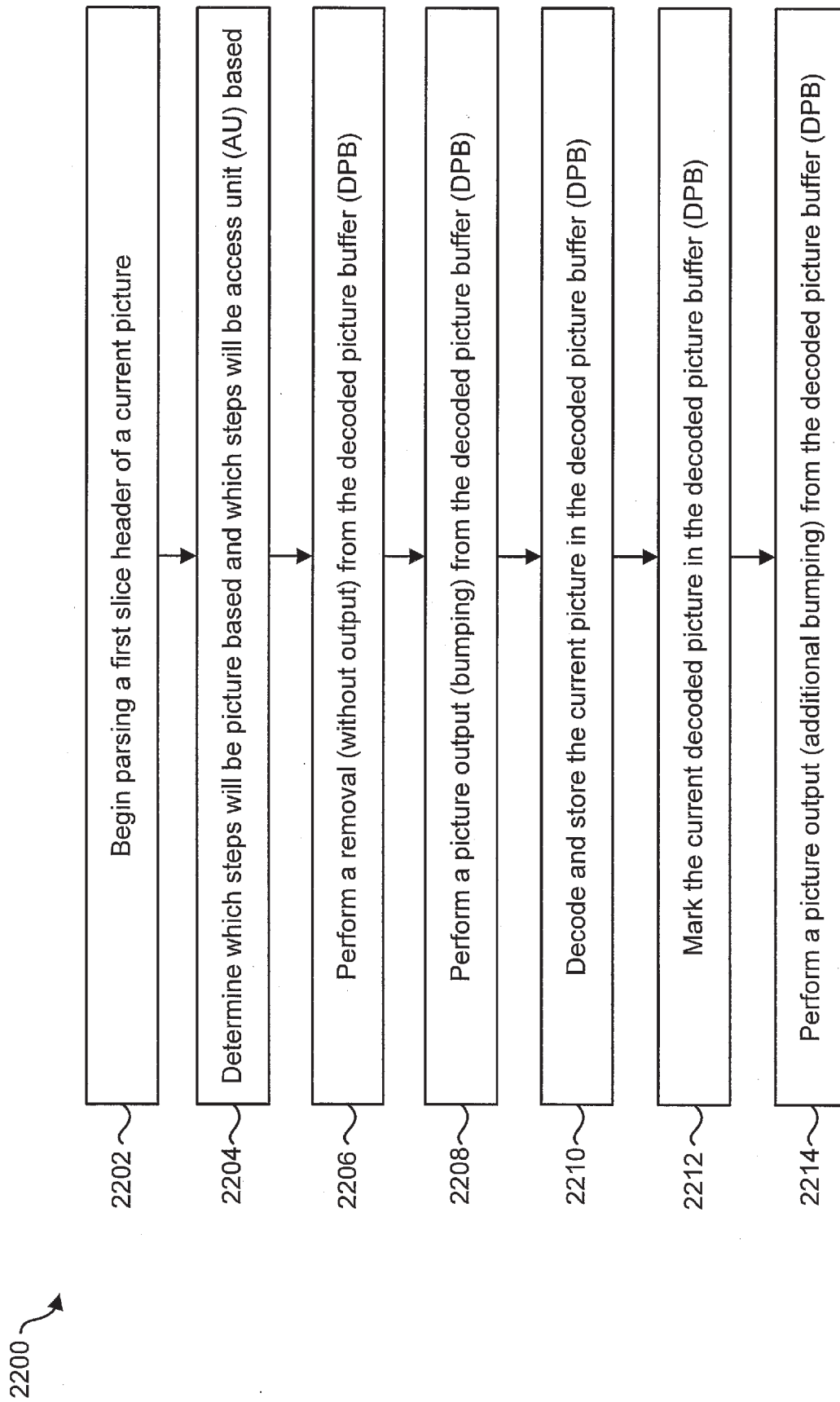
seq_parameter_set_rbsp() {	Descriptor
sps_video_parameter_set_id	u(4)
sps_max_sub_layers_minus1	u(3)
sps_temporal_id_nesting_flag	u(1)
if(nuh_layer_id == 0)	
profile_tier_level(1, sps_max_sub_layers_minus1)	
sps_seq_parameter_set_id	ue(v)
...	
log2_max_pic_order_cnt_lsb_minus4	ue(v)
if(nuh_layer_id > 0)	
sps_dpb_params_present_flag	u(1)
if((nuh_layer_id == 0) ((nuh_layer_id > 0) && sps_dpb_params_present_flag)) {	
sps_sub_layer_ordering_info_present_flag	u(1)
for(i = (sps_sub_layer_ordering_info_present_flag ? 0 : sps_max_sub_layers_minus1);	
i <= sps_max_sub_layers_minus1; i++) {	
sps_max_dec_pic_buffering_minus1[i]	ue(v)
sps_max_num_reorder_pics[i]	ue(v)
sps_max_latency_increase_plus1[i]	ue(v)
}	
}	
log2_min_luma_coding_block_size_minus3	ue(v)
log2_diff_max_min_luma_coding_block_size	ue(v)
log2_min_transform_block_size_minus2	ue(v)
...	
sps_extension_flag	u(1)
if(sps_extension_flag) {	
sps_extension()	
sps_extension2_flag	u(1)
if(sps_extension2_flag)	
while(more_rbsp_data())	
sps_extension_data_flag	u(1)
}	
rbsp_trailing_bits()	
}	

FIG. 62

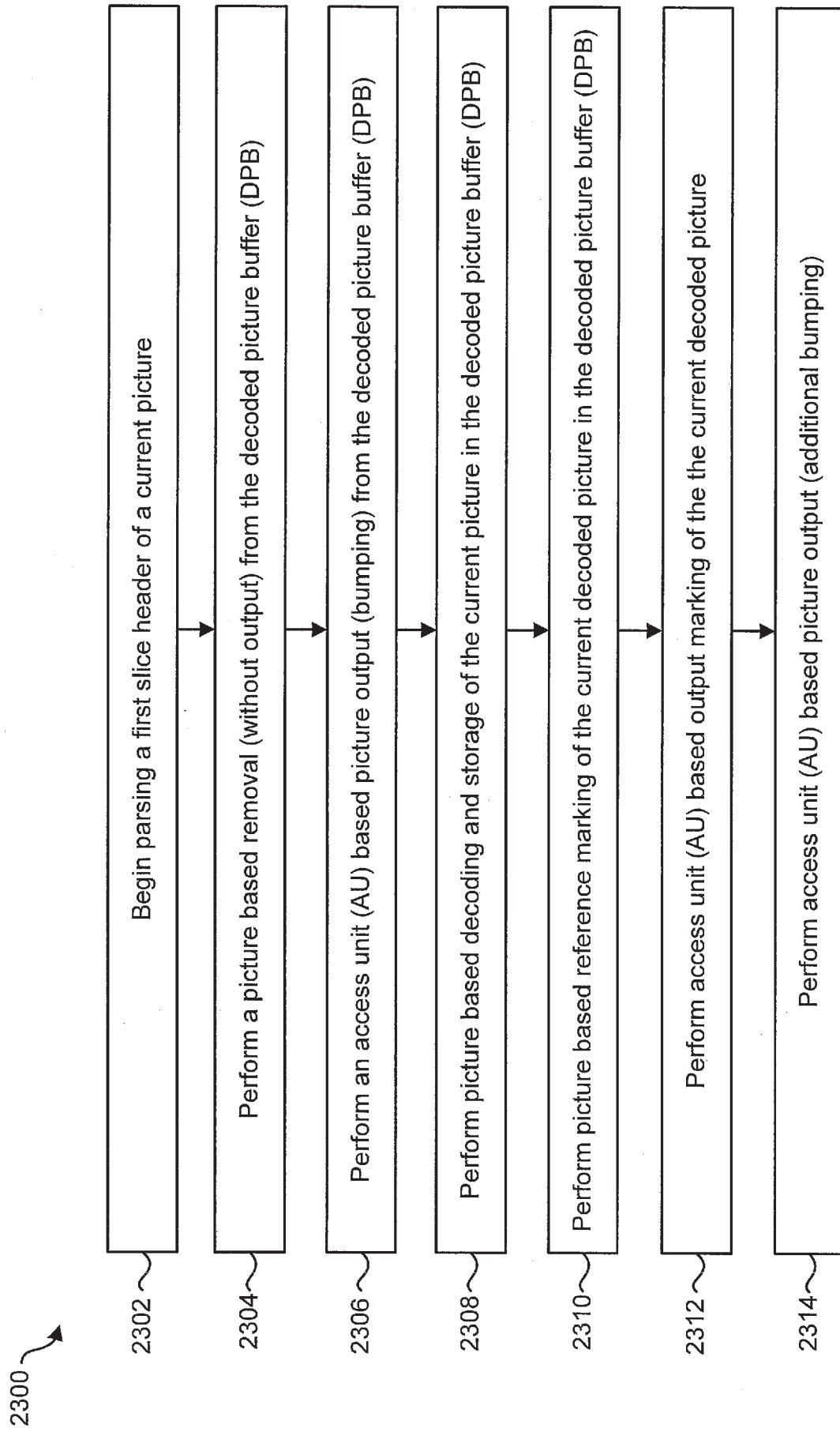
[Fig. 63]

**FIG. 63**

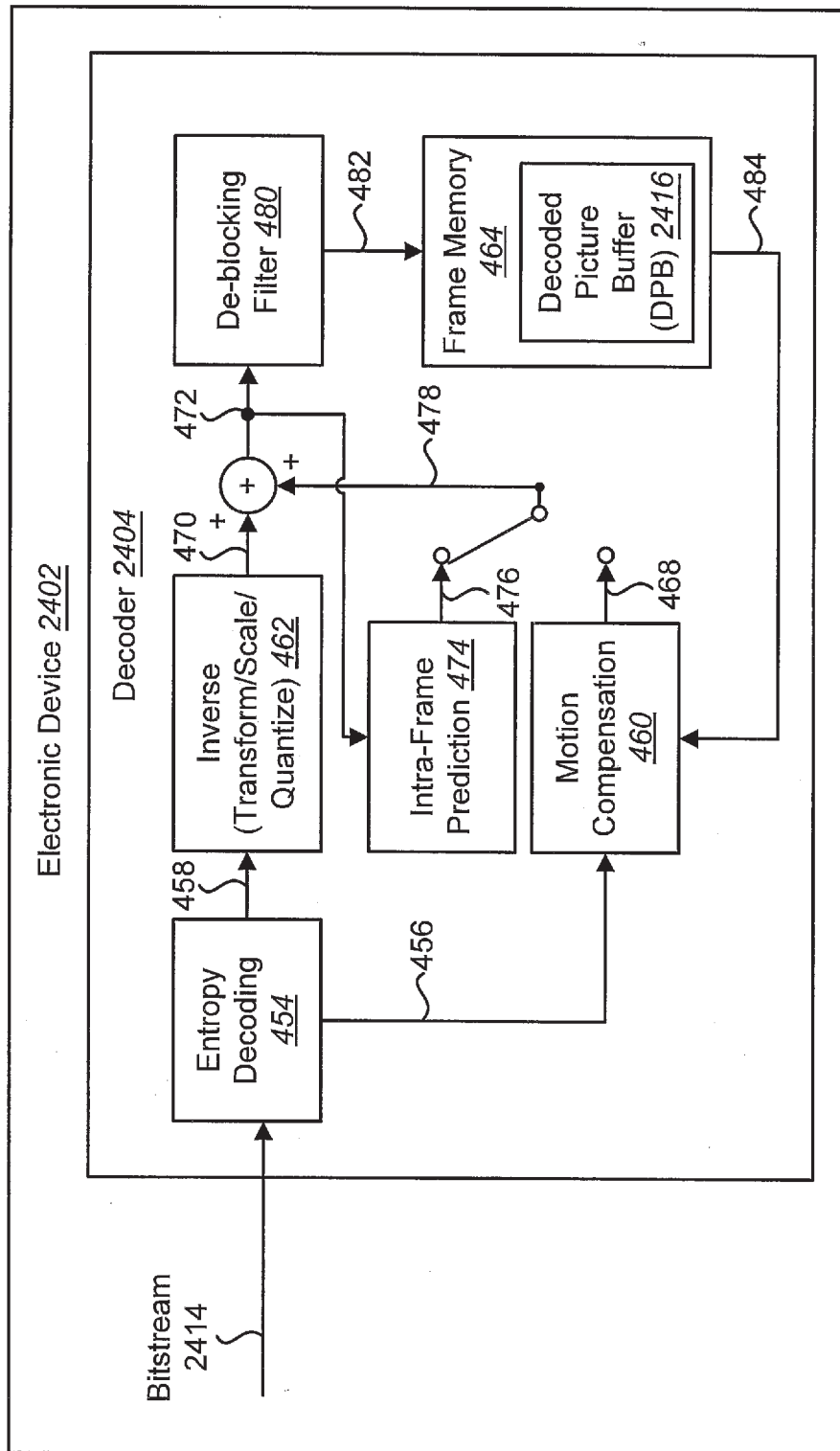
[Fig. 64]

**FIG. 64**

[Fig. 65]

**FIG. 65**

[Fig. 66]

**FIG. 66**

[Fig. 67A]

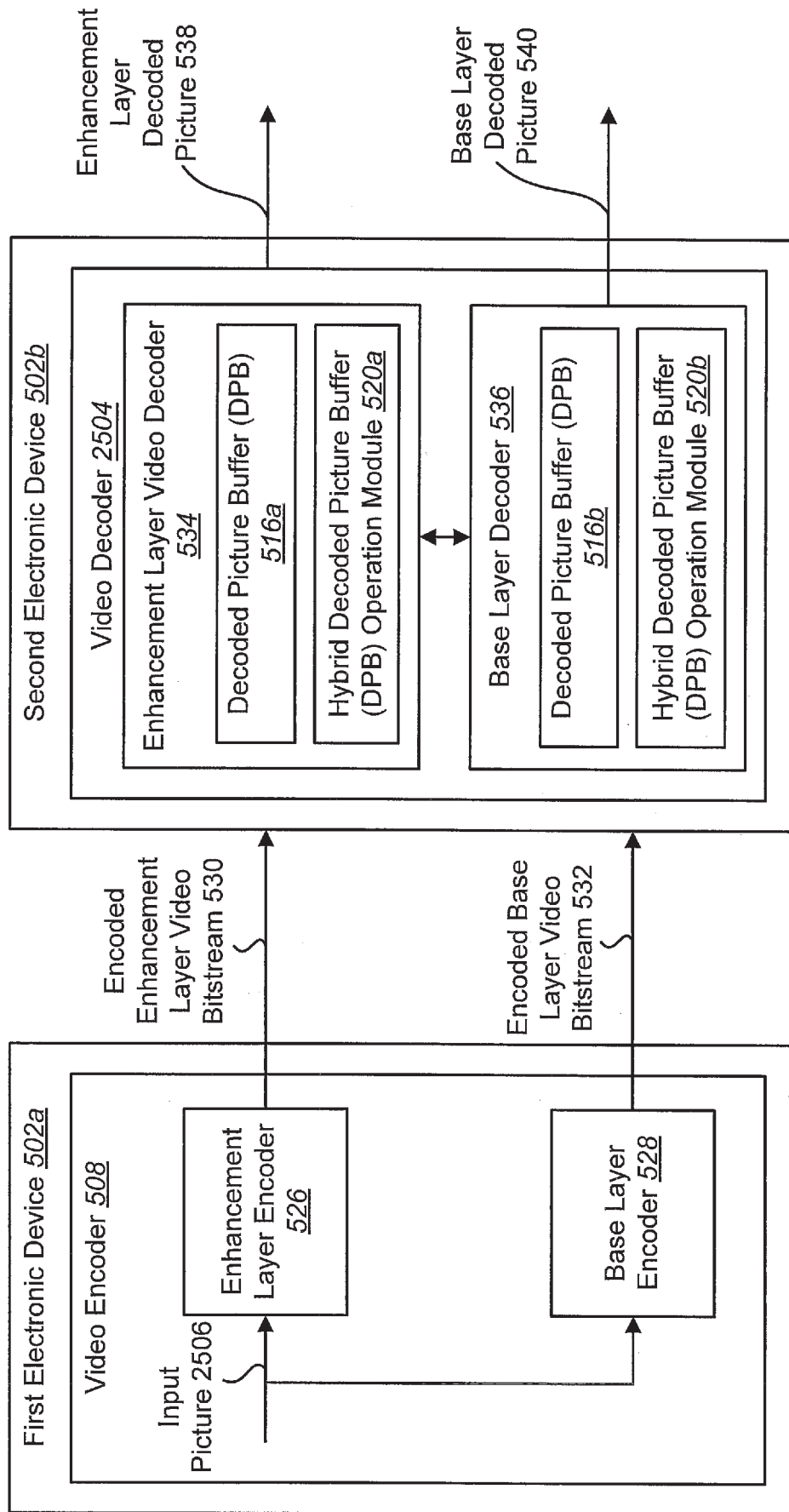
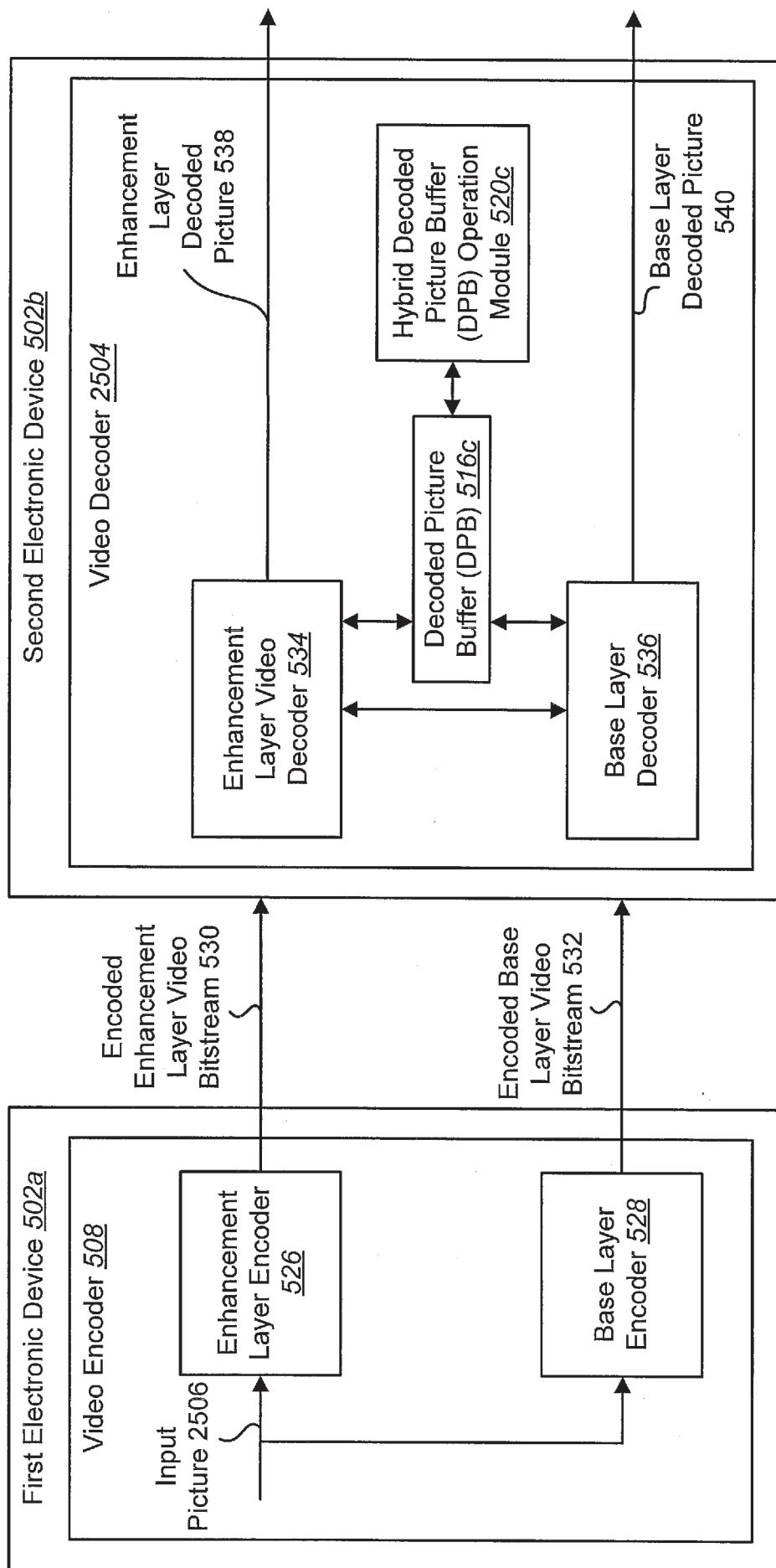
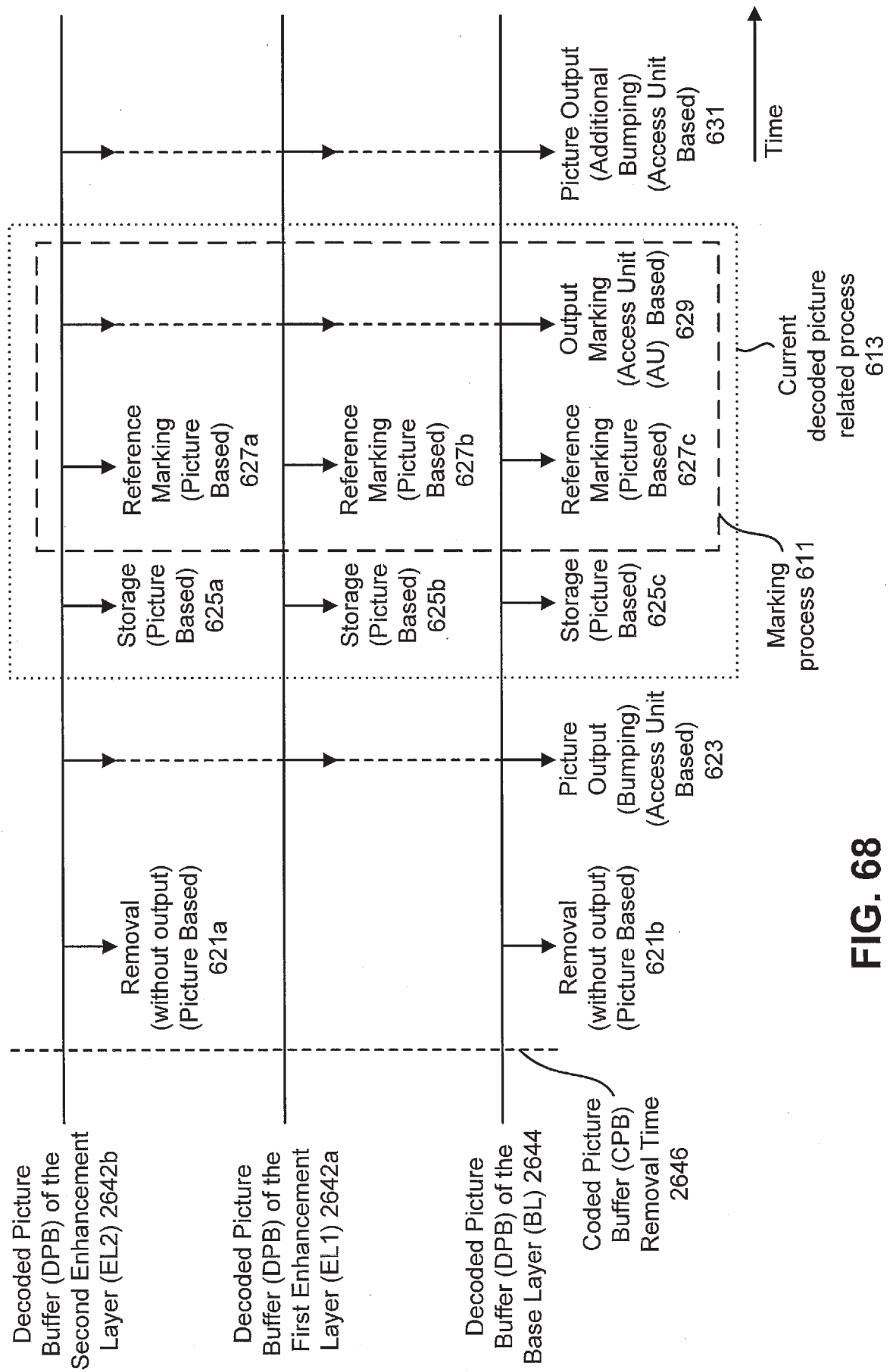


FIG. 67A

[Fig. 67B]

**FIG. 67B**

[Fig. 68]

**FIG. 68**

[Fig. 69]

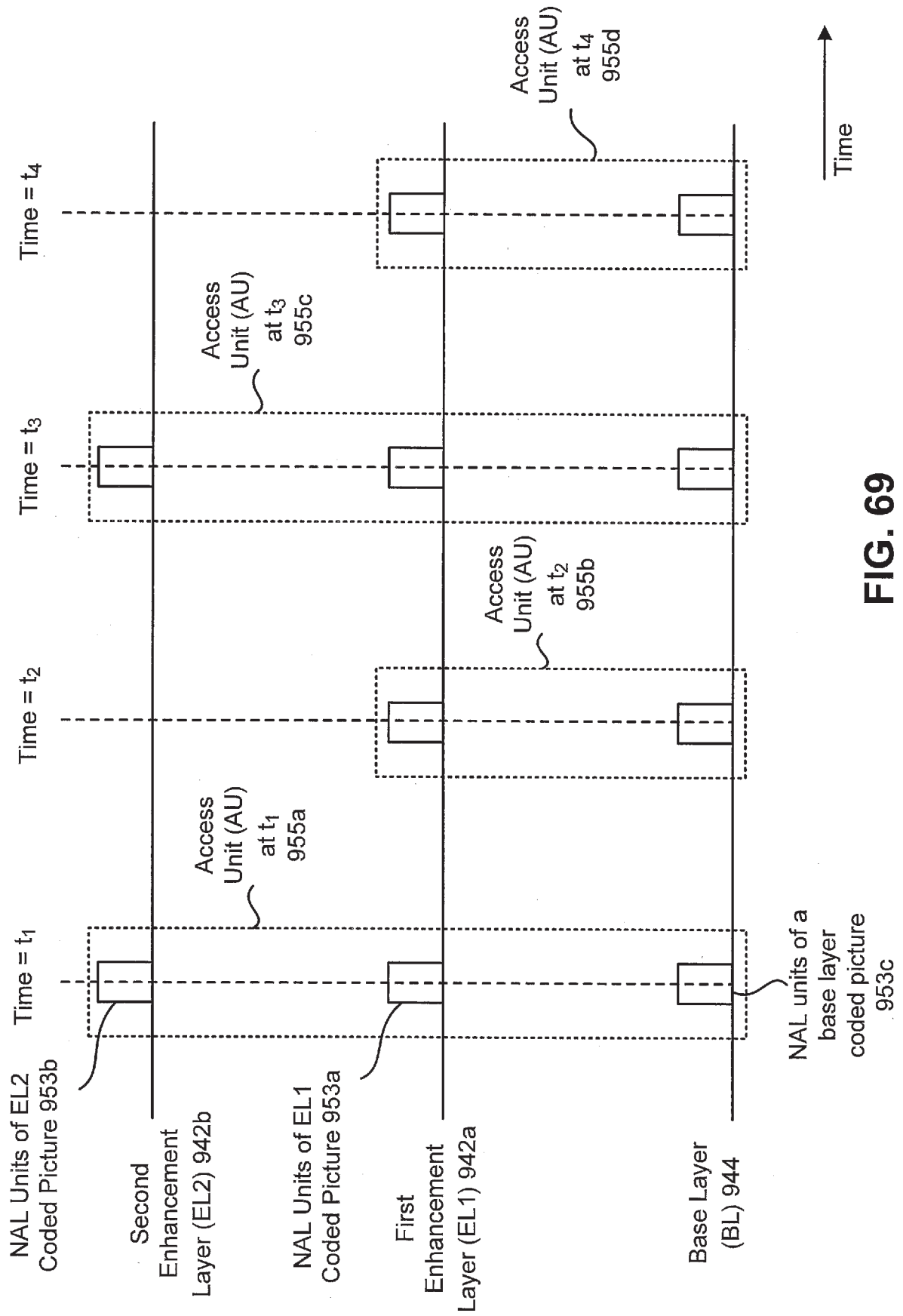


FIG. 69

[Fig. 70]

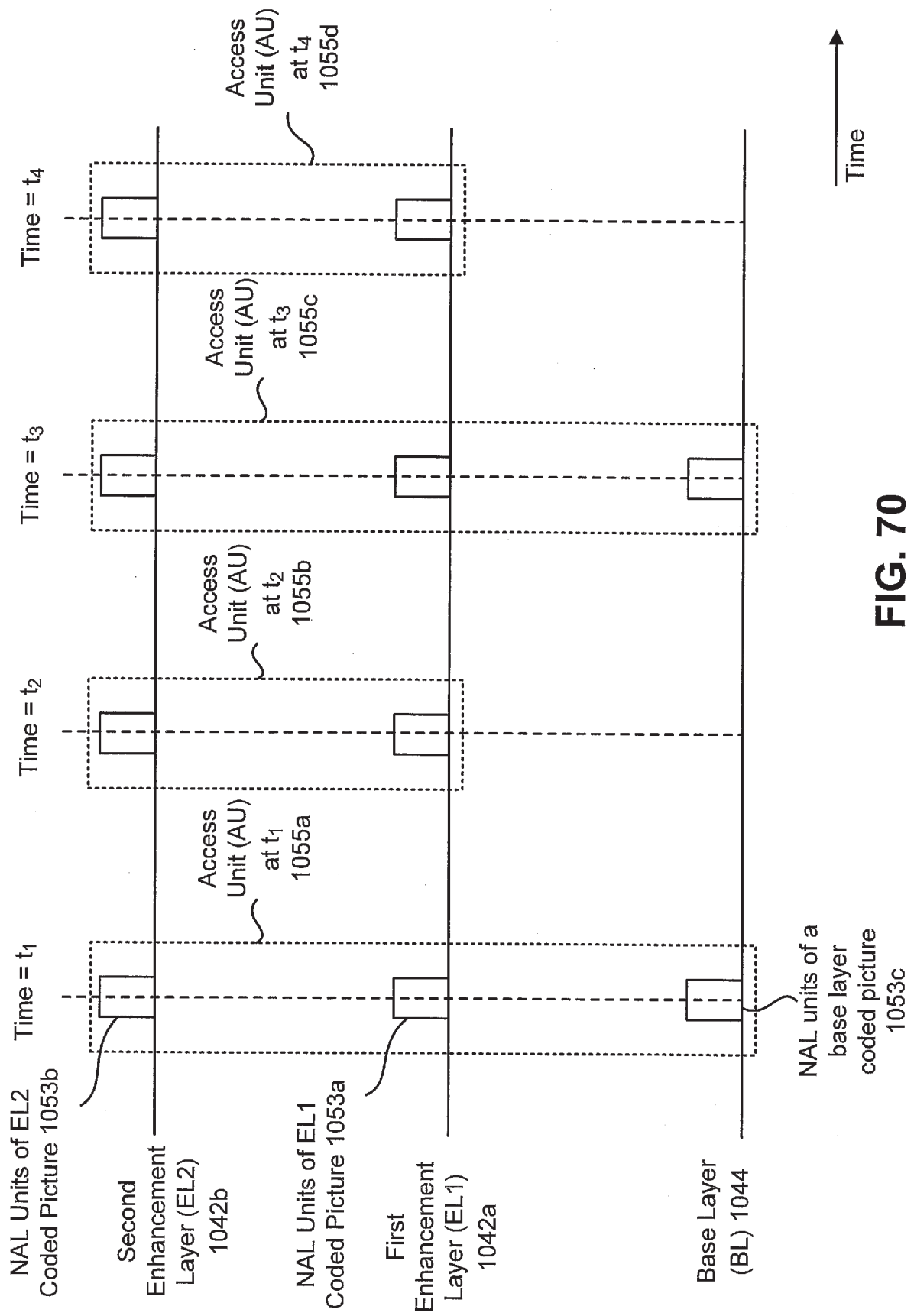


FIG. 70