

FIG. 1

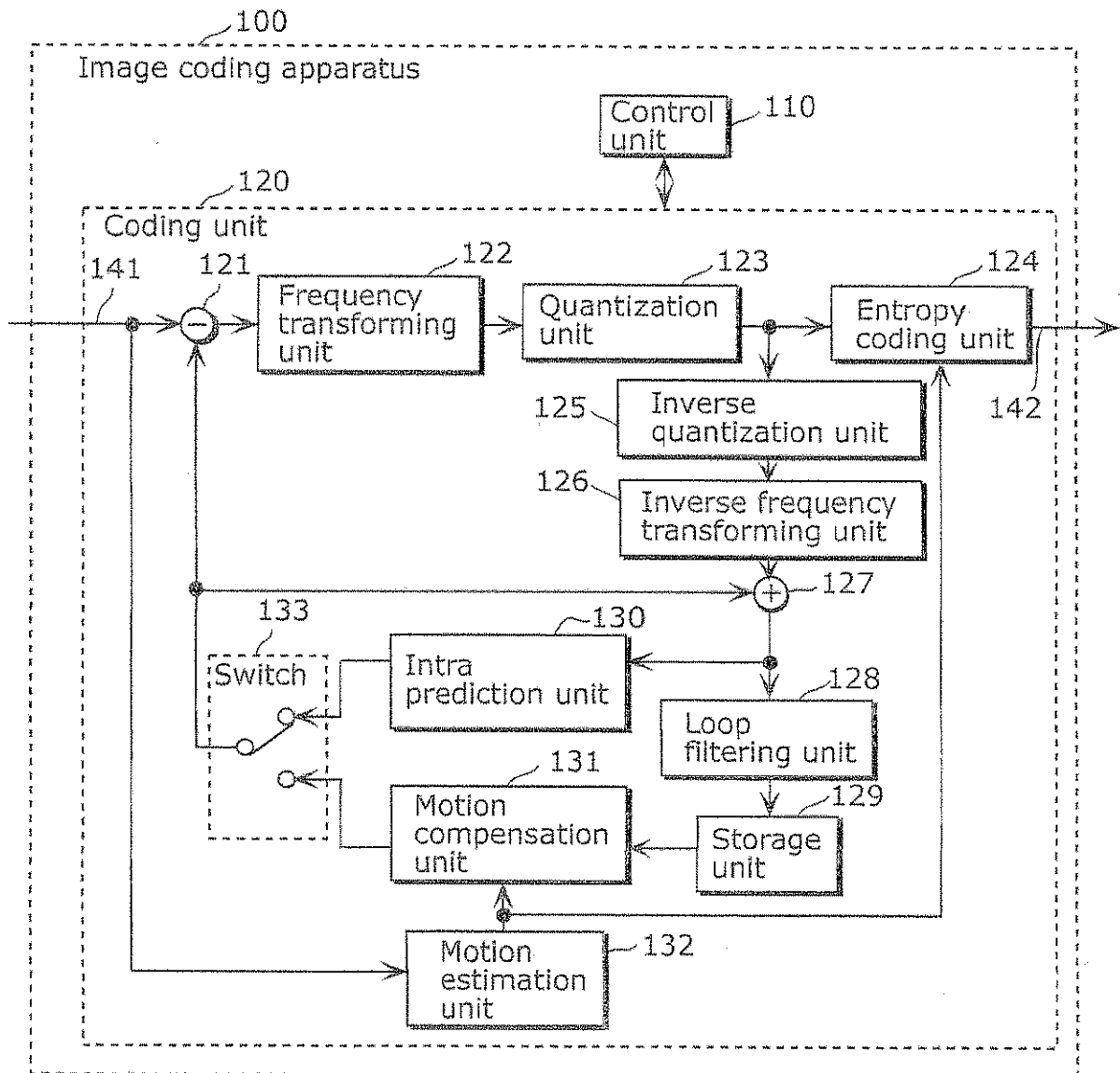


FIG. 2

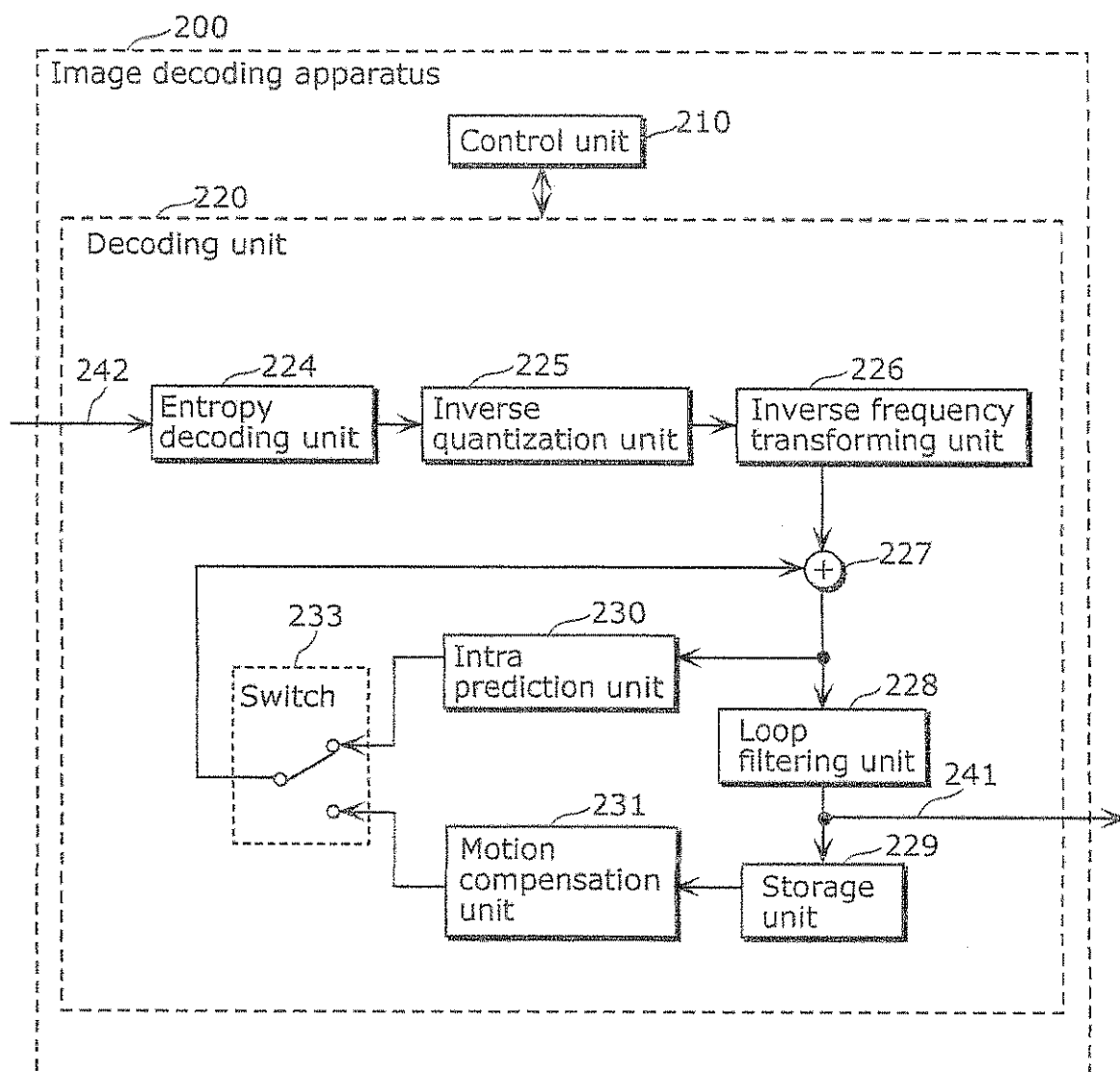


FIG. 3

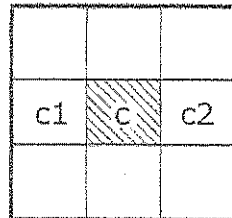


FIG. 4

Class	Condition	Edge shape
1	$c < c1 \ \&\& \ c < c2$	
2	$(c < c1 \ \&\& \ c == c2) \ \ (c == c1 \ \&\& \ c < c2)$	
3	$(c > c1 \ \&\& \ c == c2) \ \ (c == c1 \ \&\& \ c > c2)$	
4	$c > c1 \ \&\& \ c > c2$	
0	None of the above	

FIG. 5

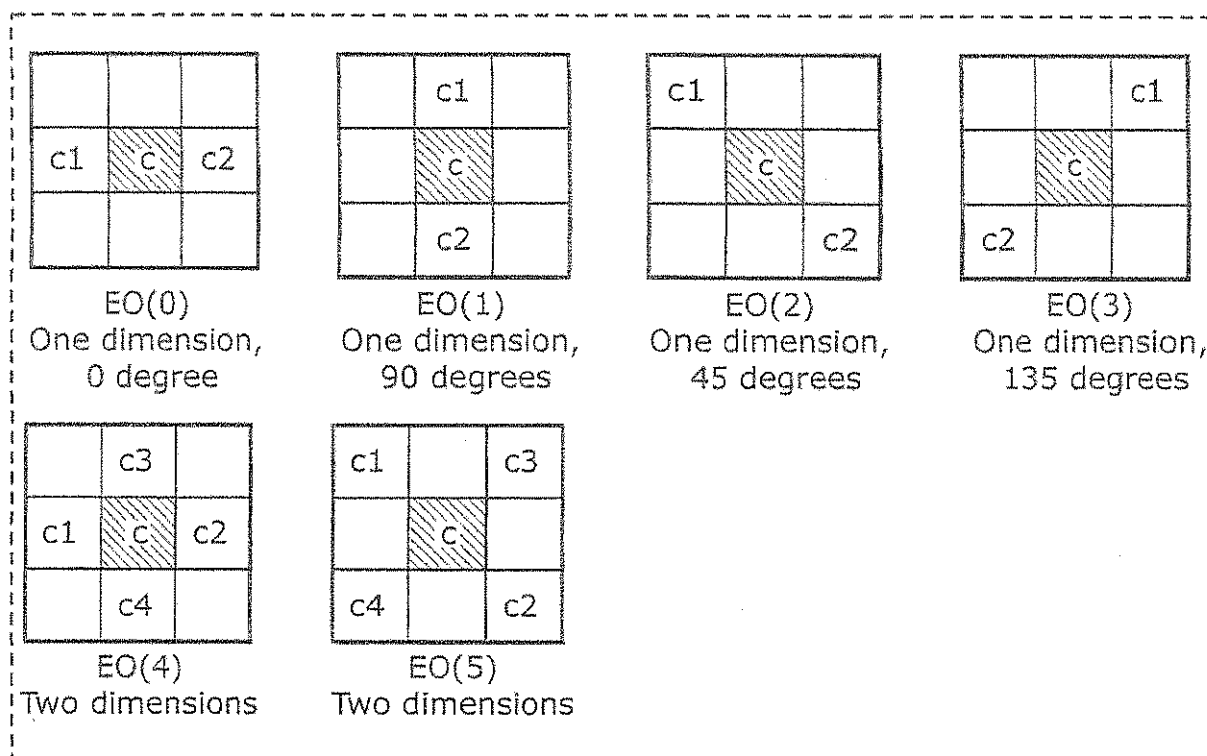


FIG. 6

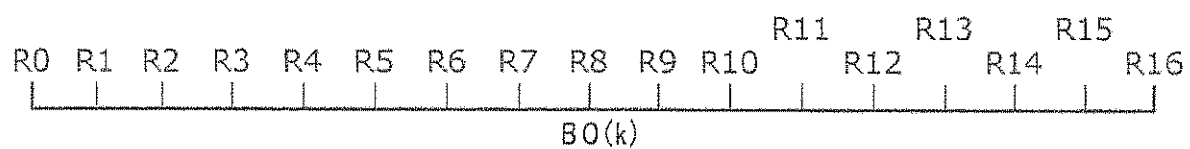


FIG. 7

Class	Conditions	Class	Conditions
1	$R0 \leq c < R1$	9	$R8 \leq c < R9$
2	$R1 \leq c < R2$	10	$R9 \leq c < R10$
3	$R2 \leq c < R3$	11	$R10 \leq c < R11$
4	$R3 \leq c < R4$	12	$R11 \leq c < R12$
5	$R4 \leq c < R5$	13	$R12 \leq c < R13$
6	$R5 \leq c < R6$	14	$R13 \leq c < R14$
7	$R6 \leq c < R7$	15	$R14 \leq c < R15$
8	$R7 \leq c < R8$	16	$R15 \leq c \leq R16$

FIG. 8

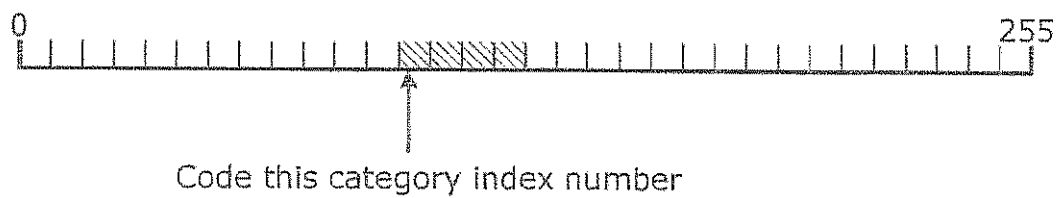


FIG. 9A

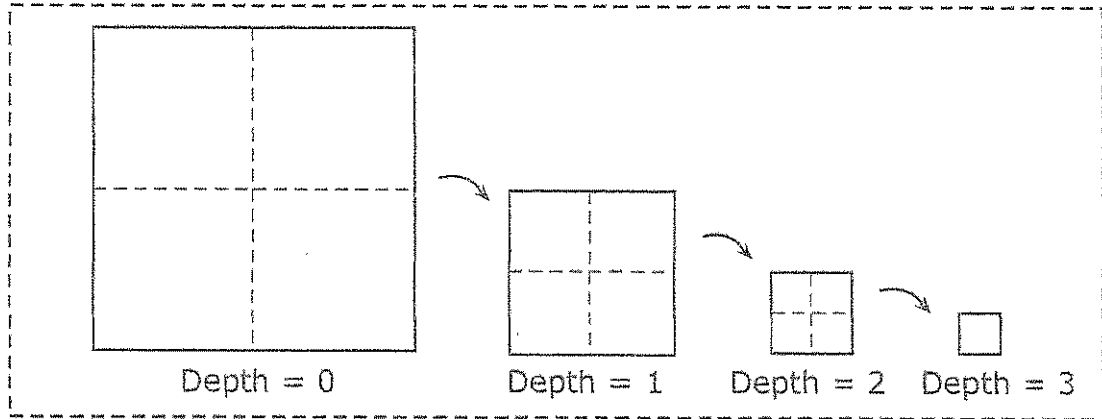


FIG. 9B

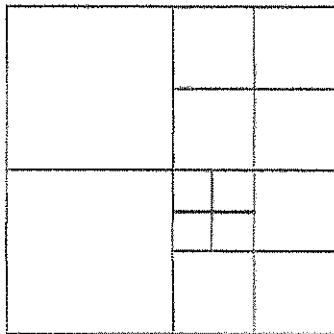


FIG. 10

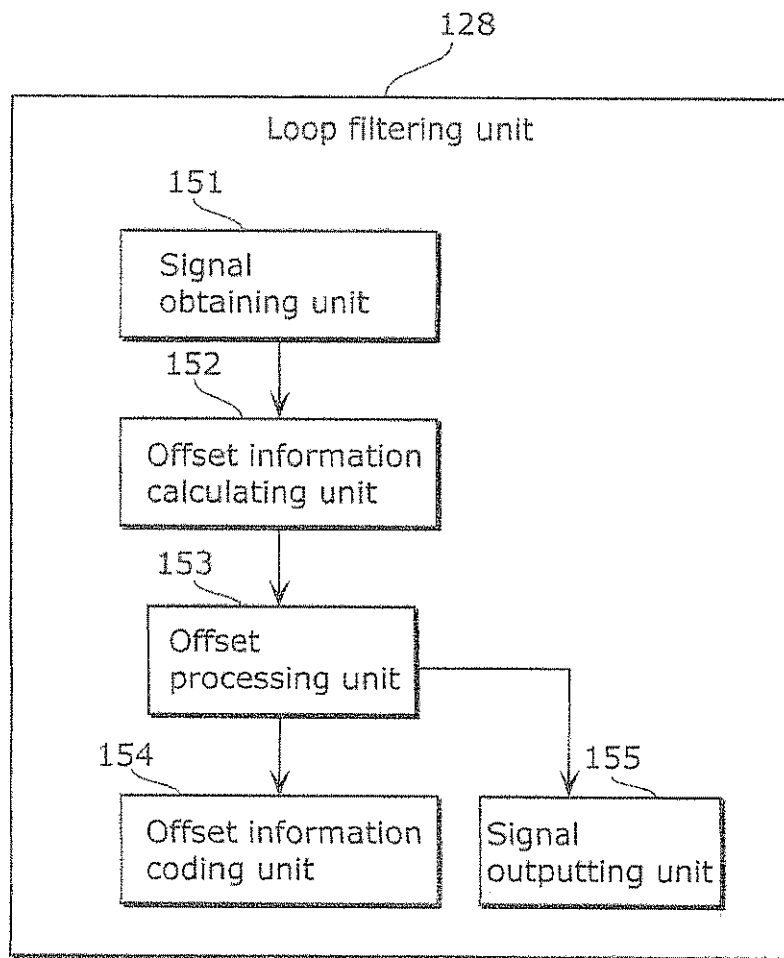


FIG. 11

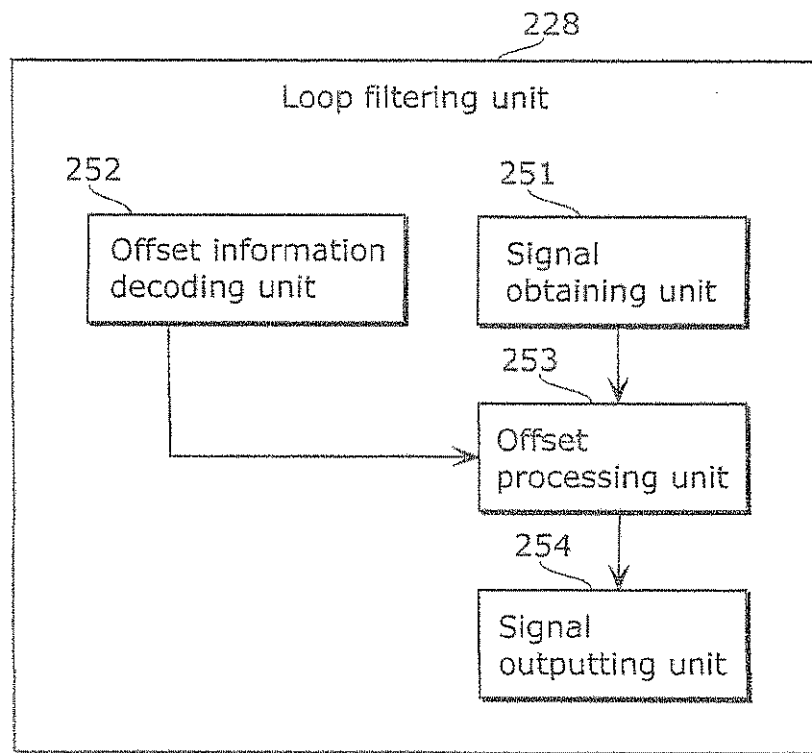


FIG. 12

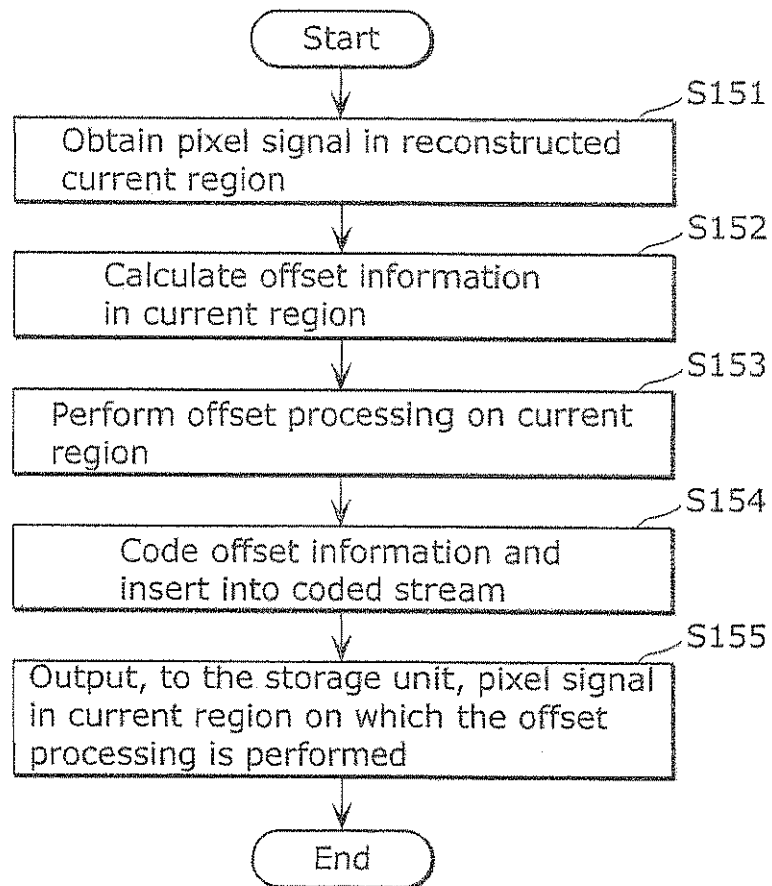


FIG. 13

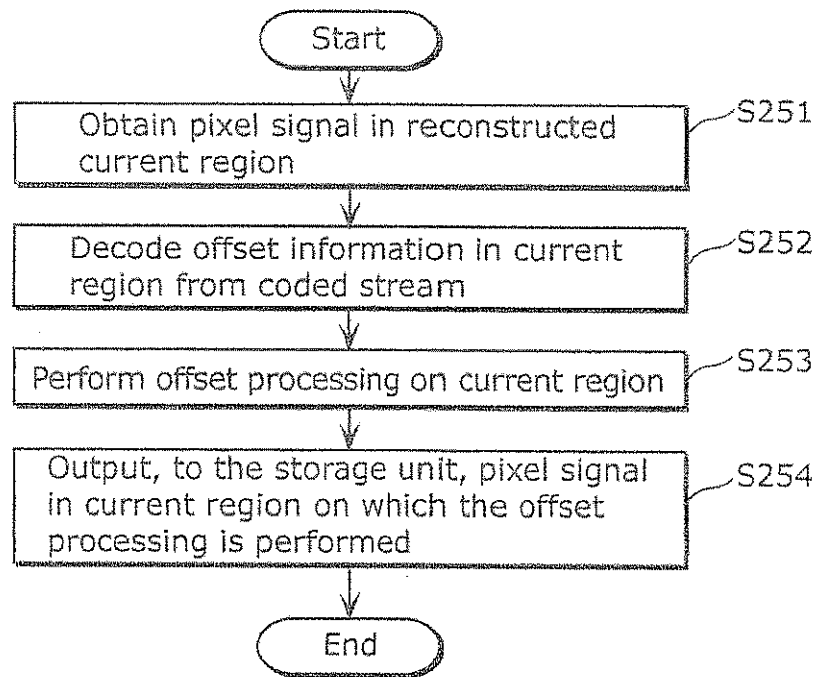


FIG. 14

Index	Bit allocation	Pixel classifying method
0	0	Do not perform offset processing
1	10	Edge offset (0)
2	110	Edge offset (1)
3	1110	Edge offset (2)
4	11110	Edge offset (3)
5	11111	Band offset

FIG. 15

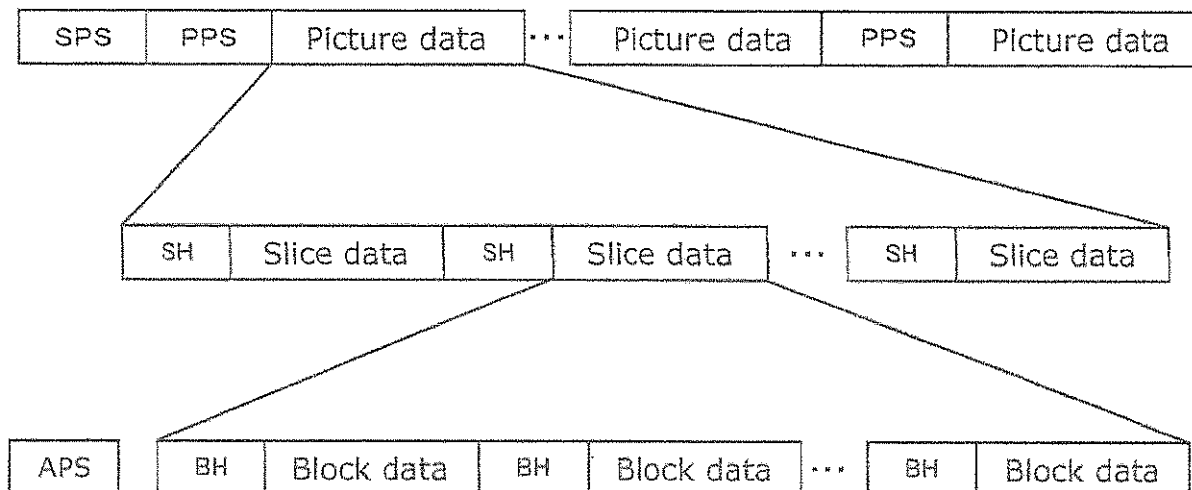


FIG. 16A

aps_sao_param(){	Descriptor
sao_cb_enable_flag	u(1)
sao_cr_enable_flag	u(1)
sao_num_lcu_in_width_minus1	ue(v)
sao_num_lcu_in_height_minus1	ue(v)
sao_one_luma_unit_flag	u(1)
if(sao_one_luma_unit_flag)	
sao_offset_vlc(0, 0, 0)	
if(sao_cb_enable_flag){	
sao_one_cb_unit_flag	u(1)
if(sao_one_cb_unit_flag)	
sao_offset_vlc(0, 0, 1)	
}	
if(sao_cr_enable_flag){	
sao_one_cr_unit_flag	u(1)
if(sao_one_cr_unit_flag)	
sao_offset_vlc(0, 0, 2)	
}	
for(ry = 0; ry <= sao_num_lcu_in_height_minus1; ry++){	
for(rx = 0; rx <= sao_num_lcu_in_width_minus1; rx++){	
if(aps_sample_adaptive_offset_flag && !sao_one_luma_unit_flag){	
if(ry > 0 && rx == 0)	
sao_repeat_row_flag[0]	u(1)
sao_unit_vlc(rx, ry, 0)	
}	
if(sao_cb_enable_flag && !sao_one_cb_unit_flag){	
if(ry > 0 && rx == 0)	
sao_repeat_row_flag[1]	u(1)
sao_unit_vlc(rx, ry, 1)	
}	
if(sao_cr_enable_flag && !sao_one_cr_unit_flag){	
if(ry > 0 && rx == 0)	
sao_repeat_row_flag[2]	u(1)
sao_unit_vlc(rx, ry, 2)	
}	
}	
}	
}	

FIG. 16B

sao_unit_vlc(rx, ry, cIdx){	Descriptor
if(!sao_repeat_row_flag[cIdx]){	
if(rx == 0 run[cIdx][rx][ry] < 0)	
if(ry == 0){	
sao_run_diff	u(v)
saoRun[cIdx][rx][ry] = sao_run_diff	
} else {	
sao_run_diff	se(v)
saoRun[cIdx][rx][ry] = sao_run_diff + saoRun[cIdx][rx][ry - 1]	
}	
saoRun[cIdx][rx + 1][ry] = saoRun[cIdx][rx][ry] - 1	
if(rx == 0 saoRun[cIdx][rx][ry] < 0)	
if(ry > 0)	
sao_merge_up_flag	u(1)
if(!sao_merge_up_flag)	
sao_offset_vlc(rx, ry, cIdx)	
} else	
saoRun[cIdx][rx][ry] = saoRun[cIdx][rx][ry - 1]	
}	

FIG. 16C

sao_offset_vlc(rx, ry, cIdx){	Descriptor
sao_type_idx [cIdx][rx][ry]	ue(v)
if(sao_type_idx[cIdx][rx][ry] == 5){	
sao_band_position [cIdx][rx][ry]	u(5)
for(i = 0; i < 4; i++)	
sao_offset [cIdx][rx][ry][i]	se(v)
} else if(sao_type_idx[cIdx][rx][ry] != 0)	
for(i = 0; i < 4; i++)	
sao_offset [cIdx][rx][ry][i]	ue(v)
}	

FIG. 17A

slice_data(){	Descriptor
CtbAddrRS = SliceCtbAddrRS	
CtbAddrTs = CtbAddrRStoTS[CtbAddrRs]	
moreDataFlag = 1	
if(adaptive_loop_filter_flag && alf_cu_control_flag)	
AlfCuFlagIdx = -1	
do {	
xCtb = InverseRasterScan(CtbAddrRS, CtbSize, CtbSize,	
pic_width_in_luma_samples, 0)	
yCtb = InverseRasterScan(CtbAddrRS, CtbSize, CtbSize,	
pic_width_in_luma_samples, 1)	
NumPCMBBlock = 0	
CtbAddrInSlice = CtbAddrRS - (slice_address >> SliceGranularity)	
AddrUp = CtbAddrRS - PicWidthInCtbs	
if(slice_sao_interleaving_flag){	
if(slice_sample_adaptive_offset_flag)	
sao_unit_cabac(xCtb, yCtb, 0)	
if(sao_cb_enable_flag)	
sao_unit_cabac(xCtb, yCtb, 1)	
if(sao_cr_enable_flag)	
sao_unit_cabac(xCtb, yCtb, 2)	
}	
moreDataFlag = coding_tree(xCtb, yCtb, Log2CtbSize, 0)	
CtbAddrTS++	
if(moreDataFlag && ((tiles_or_entropy_coding_sync_idc == 1 &&	
TileId[CtbAddrTS] != TileId[CtbAddrTS - 1])	
(tiles_or_entropy_coding_sync_idc == 2 &&	
num_substream_minus1 > 0 &&	
CtbAddrTs/PicWidthInCtbs <= num_substream_minus1 &&	
CtbAddrTS % PicWidthInCtbs == 0))) {	
rbsp_trailing_bits()	
if(nextbits(24) == 0x0000002){	
entry_point_marker_two_3bytes	f(24)
tile_idx_minus_1	u(v)
}	
}	
} while(moreDataFlag)	
}	

FIG. 17B

	Descriptor
sao_unit_cabac(rx, ry, cIdx){	
if(rx > 0) {	
if(CtbAddrInSlice!=0 &&	
TileId[CtbAddrTS] == TileId[CtbAddrRStoTS[CtbAddrRS - 1]])	
sao_merge_left_flag	ae(v)
}	
if(!sao_merge_left_flag) {	
if(ry > 0) {	
if(((CtbAddrTS - CtbAddrRStoTS[CtbAddrRS-PicWidthInCtbs])	
<=CtbAddrInSlice) &&	
(TileId[CtbAddrTS]==	
TileId[CtbAddrRStoTS[CtbAddrRS- PicWidthInCtbs]]))	
sao_merge_up_flag	ae(v)
}	
if(!sao_merge_up_flag)	
sao_offset_cabac(rx, ry, cIdx)	
}	
}	

FIG. 17C

	Descriptor
sao_offset_cabac(rx, ry, cIdx) {	
sao_type_idx [cIdx][rx][ry]	ae(v)
if(sao_type_idx[cIdx][rx][ry] == 5)	
sao_band_position [cIdx][rx][ry]	ae(v)
if(sao_type_idx[cIdx][rx][ry] != 0) {	
for(i = 0; i < 4; i++)	
sao_offset [cIdx][rx][ry][i]	ae(v)
}	
if(sao_type_idx[cIdx][rx][ry] == 5) {	
for(i = 0; i < 4; i++) {	
if(sao_offset[cIdx][rx][ry] != 0)	
sao_offset_sign [cIdx][rx][ry][i]	ae(v)
}	
}	
}	

FIG. 18

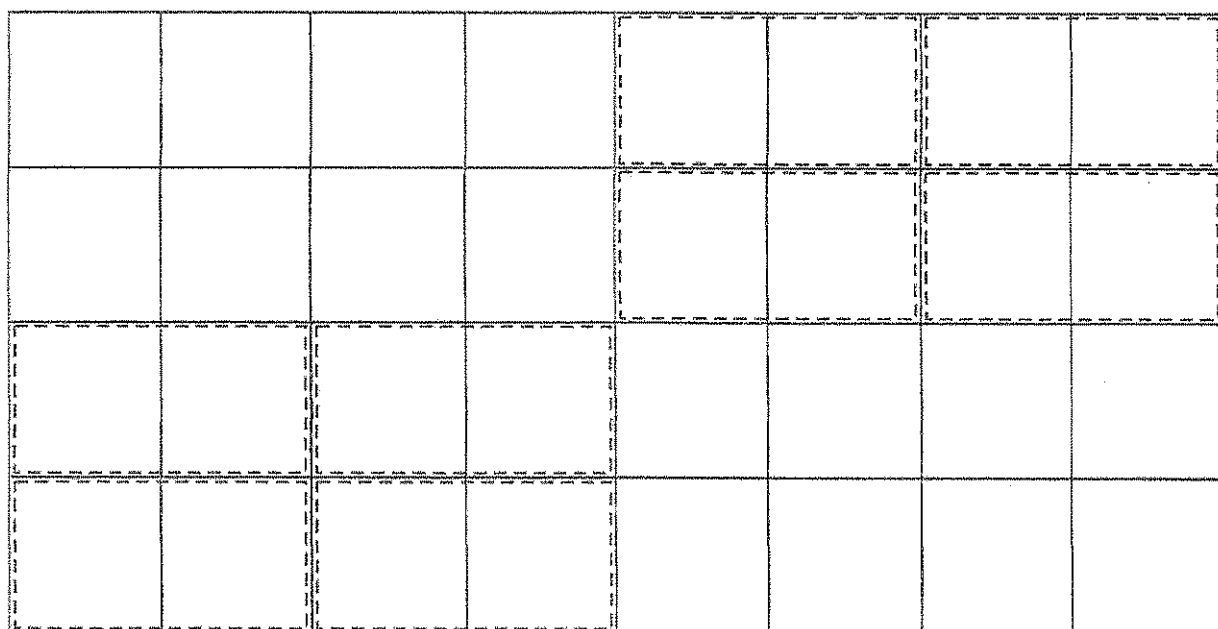


FIG. 19

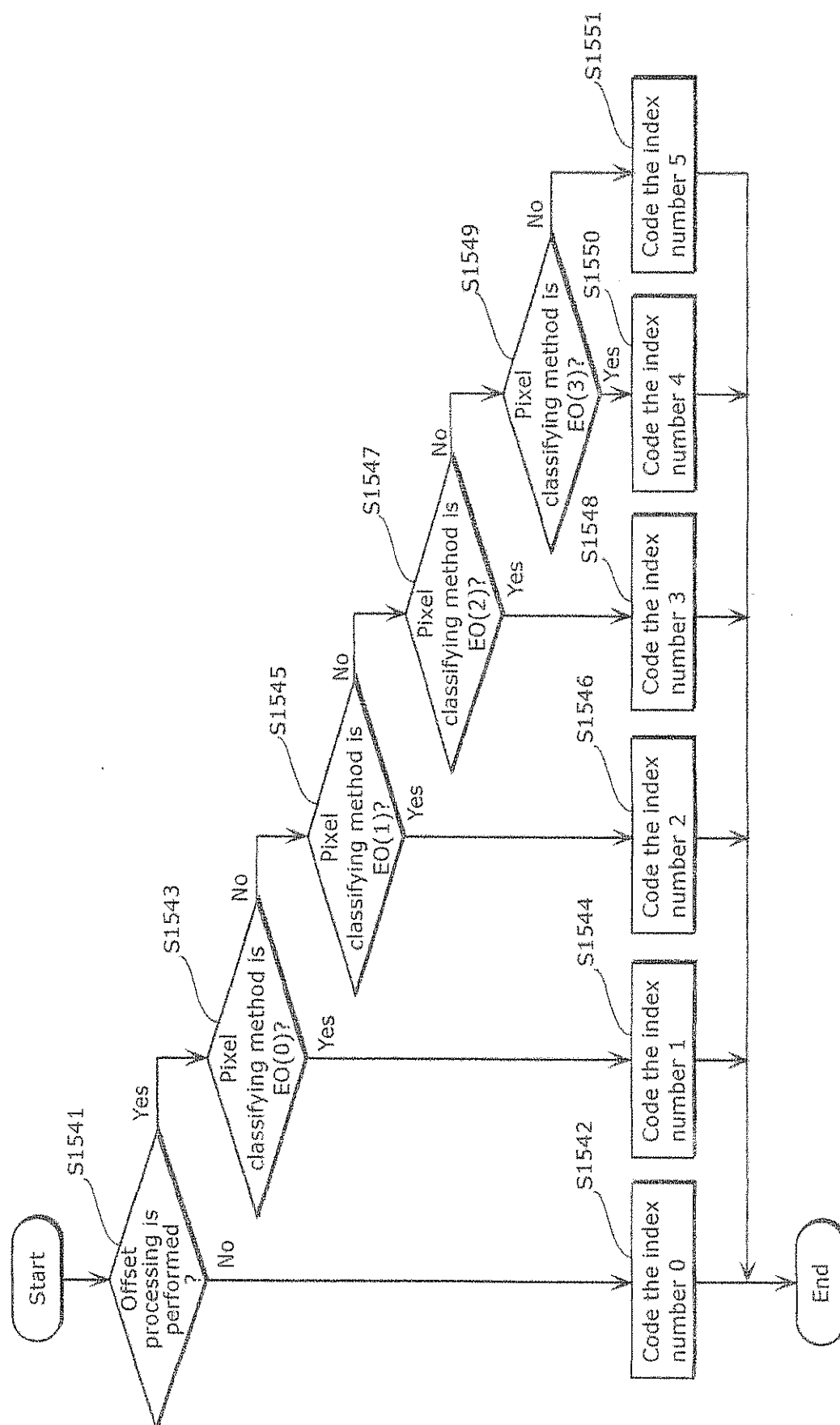


FIG. 20

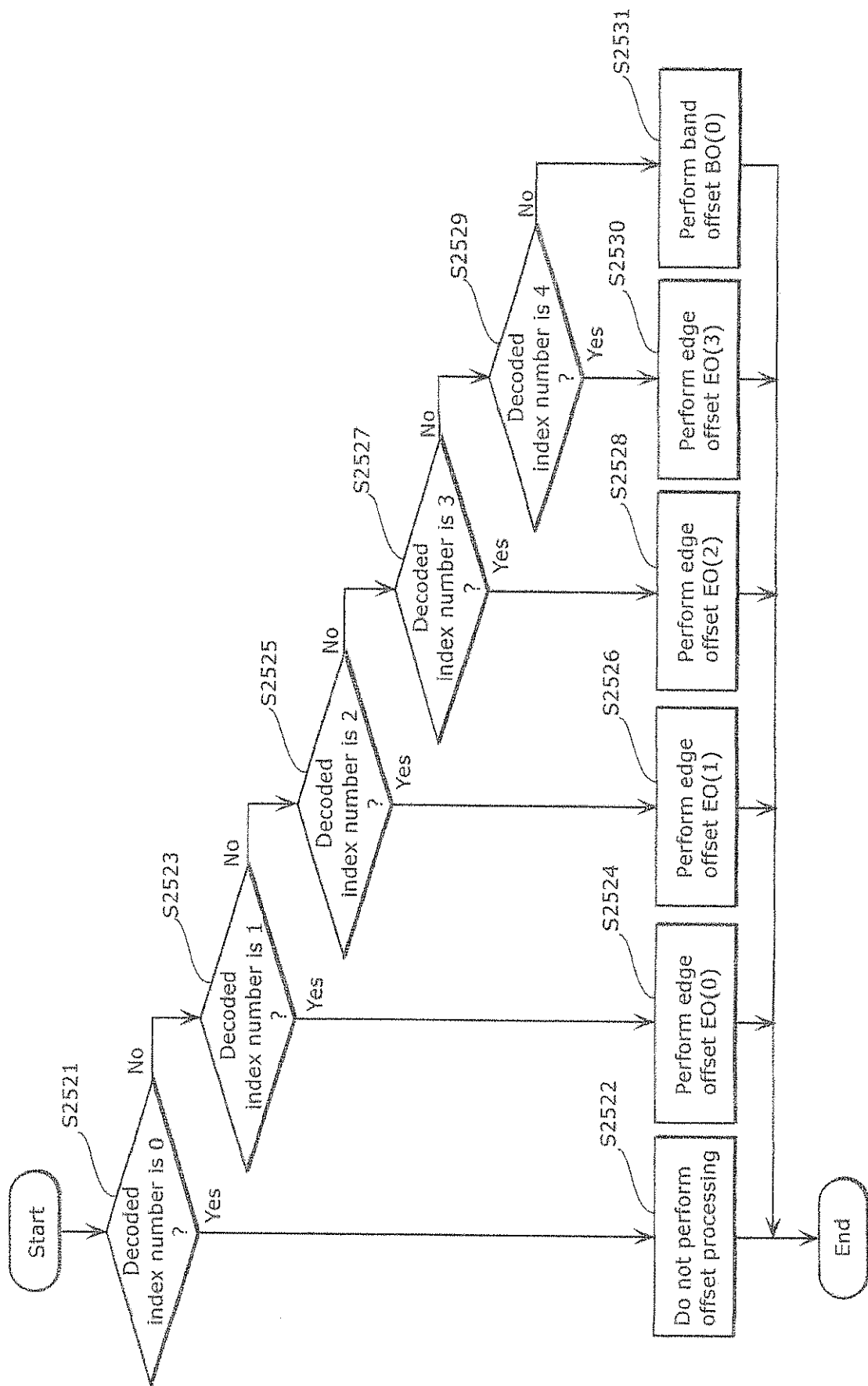


FIG. 21

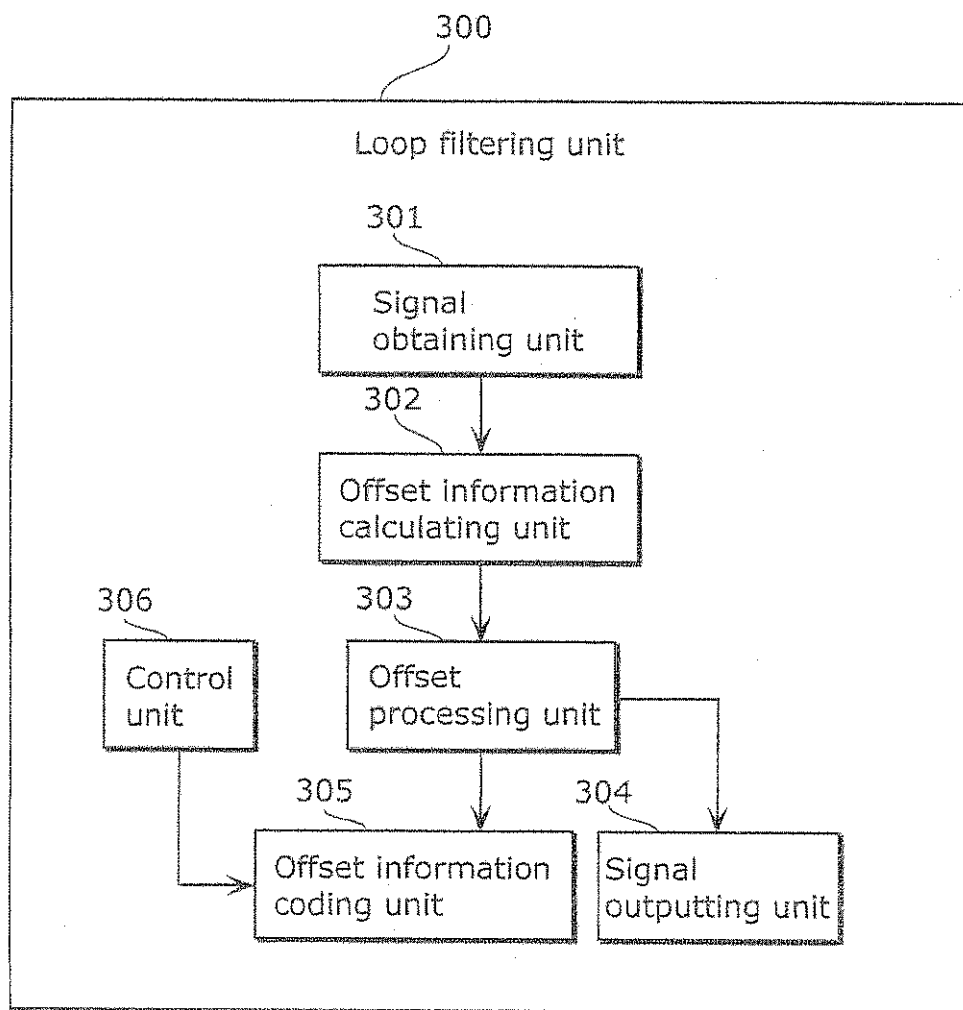


FIG. 22

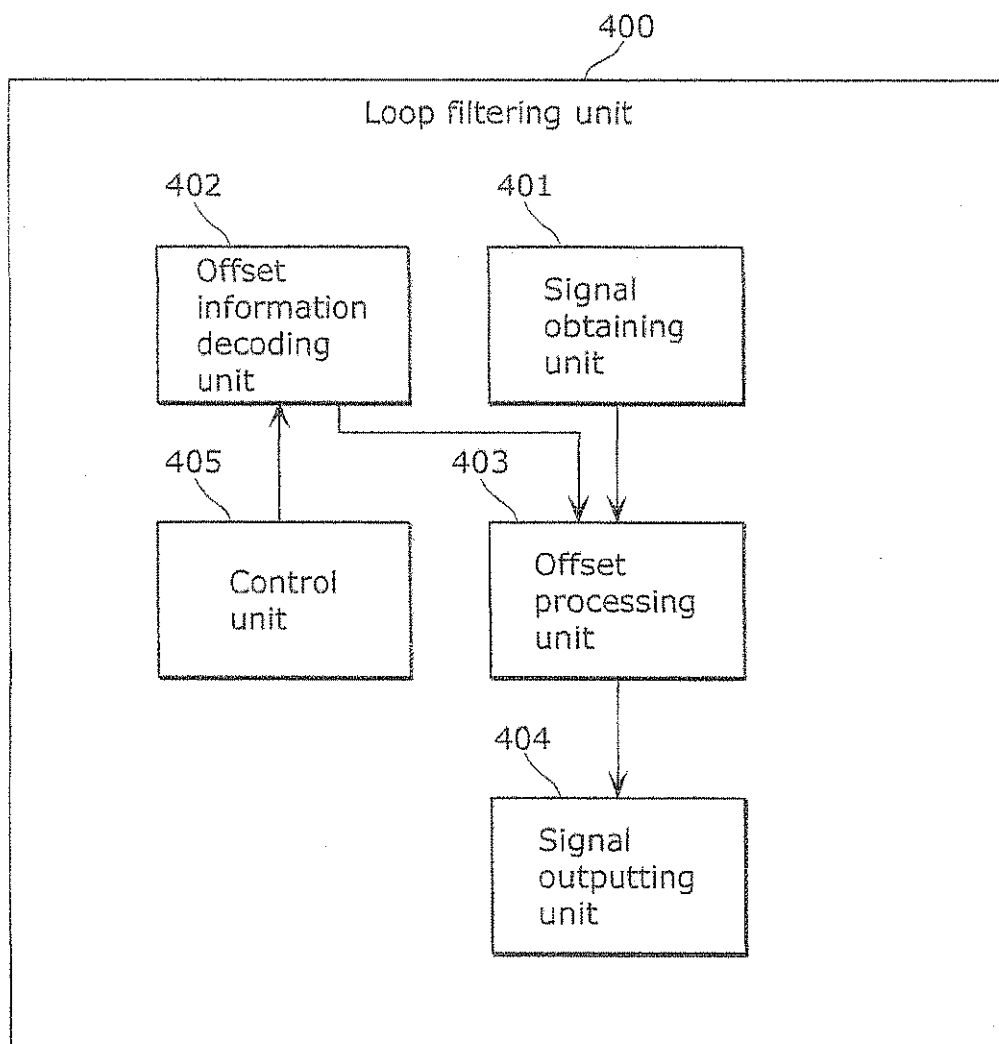


FIG. 23

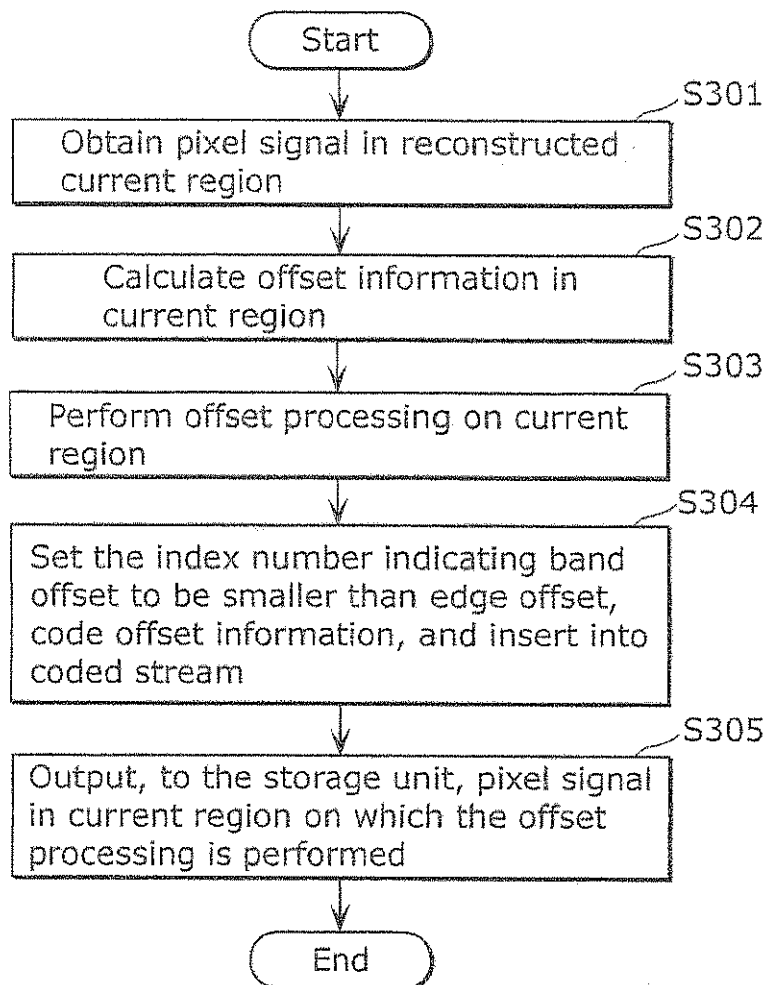


FIG. 24

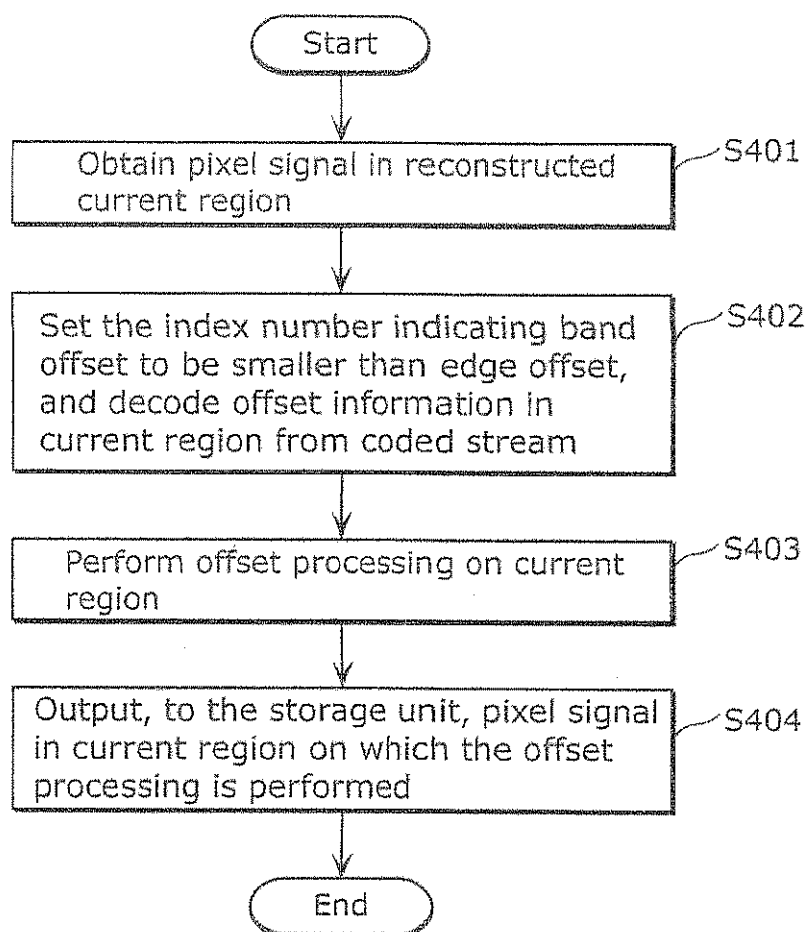


FIG. 25

Index	Bit allocation	Pixel classifying method
0	0	No offset processing
1	10	Band offset
2	110	Edge offset (0)
3	1110	Edge offset (1)
4	11110	Edge offset (2)
5	11111	Edge offset (3)

FIG. 26

Index	Bit allocation	Offset processing flag
0	0	Do not perform offset processing
1	1	Perform offset processing

Index	Bit allocation	Pixel classifying method
0	0	Band offset
1	10	Edge offset (0)
2	110	Edge offset (1)
3	1110	Edge offset (2)
4	1111	Edge offset (3)

FIG. 27A

sao_unit_vlc(rx, ry, cIdx) {	Descriptor
if(!sao_repeat_row_flag[cIdx]) {	
if(rx == 0 run[cIdx][rx][ry] < 0)	
if(ry == 0) {	
sao_run_diff	u(v)
saoRun[cIdx][rx][ry] = sao_run_diff	
} else {	
sao_run_diff	se(v)
saoRun[cIdx][rx][ry] = sao_run_diff + saoRun[cIdx][rx][ry - 1]	
}	
saoRun[cIdx][rx + 1][ry] = saoRun[cIdx][rx][ry] - 1	
if(rx == 0 saoRun[cIdx][rx][ry] < 0)	
sao_on_flag ————— Offset processing flag	u(1)
if (sao_on_flag) {	
if (ry > 0)	
sao_merge_up_flag	u(1)
if(!sao_merge_up_flag)	
sao_offset_vlc(rx, ry, cIdx)	
}	
} else	
saoRun[cIdx][rx][ry] = saoRun[cIdx][rx][ry - 1]	
}	

FIG. 27B

sao_offset_vlc(rx, ry, cIdx) {	Pixel classifying	Descriptor
sao_type_idx[cIdx][rx][ry] ————— method		ue(v)
if(sao_type_idx[cIdx][rx][ry] == 5) {	Index number	
sao_band_position [cIdx][rx][ry]		u(5)
for(i = 0; i < 4; i++)		
sao_band_offset [cIdx][rx][ry][i]		se(v)
} else		
for(i = 0; i < 4; i++)		
sao_edge_offset [cIdx][rx][ry][i]		ue(v)
}		

FIG. 27C

Syntax element	ctxIdxTable, ctxIdxOffset	binIdx				
		0	1	2	3	≥ 4
sao_on_flag	Table 9-5'	0	cIdx	na	na	na
		3	cIdx	na	na	na
		6	cIdx	na	na	na

FIG. 27D

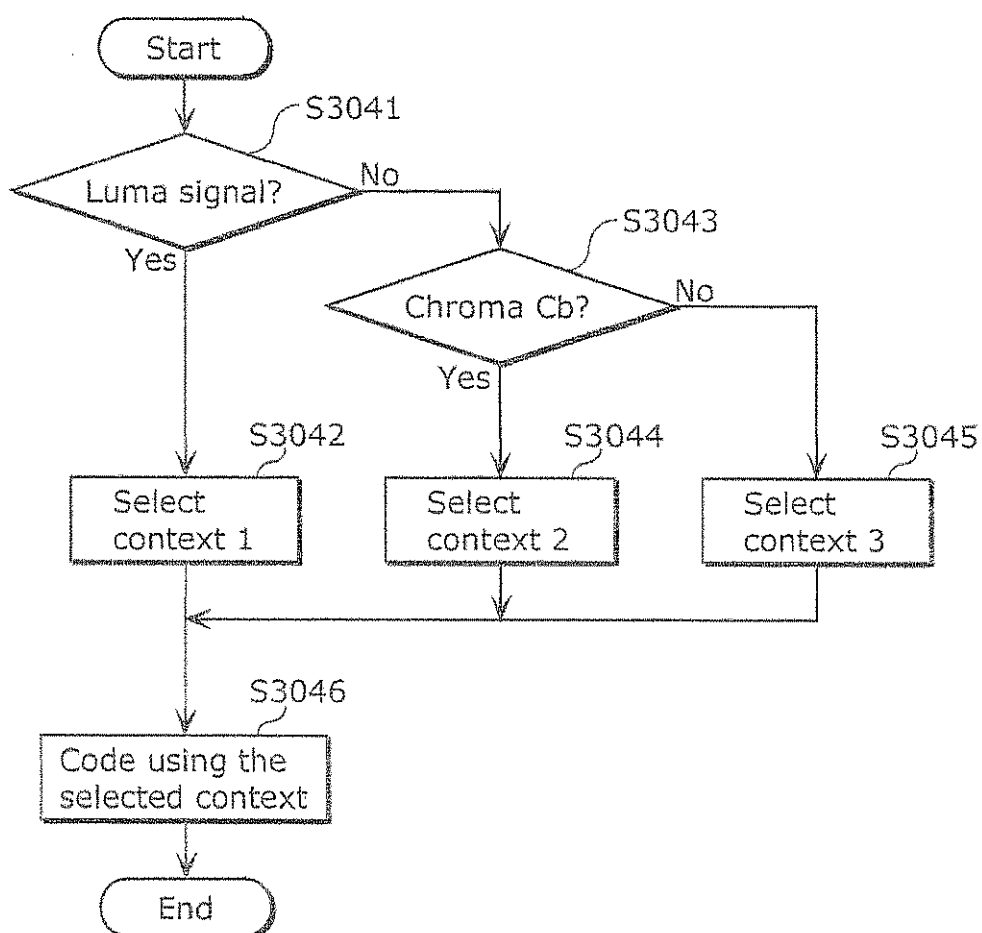


FIG. 27E

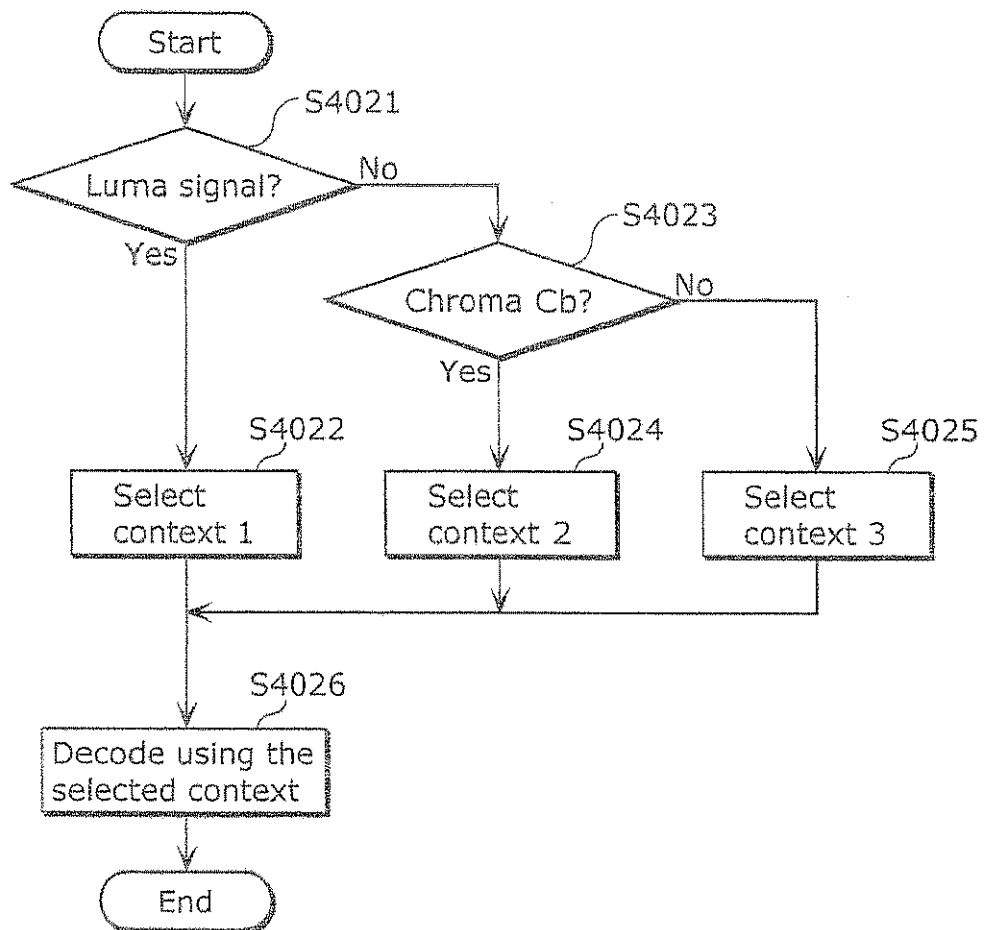


FIG. 28A

		Descriptor
sao_unit_cabac(rx, ry, cIdx){		
sao_on_flag	Offset processing flag	u(1)
if(sao_on_flag) {		
if(rx > 0) {		
if(CtbAddrInSlice != 0 &&		
TileId[CtbAddrTS] == TileId[CtbAddrRStoTS[CtbAddrRS - 1]])		
sao_merge_left_flag		ae(v)
}		
if(!sao_merge_left_flag) {		
if(ry > 0) {		
if(((CtbAddrTS - CtbAddrRStoTS[CtbAddrRS - PicWidthInCtbs])		
<= CtbAddrInSlice) &&		
(TileId[CtbAddrTS] ==		
TileId[CtbAddrRStoTS[CtbAddrRS - PicWidthInCtbs]]))		
sao_merge_up_flag		ae(v)
}		
if(!sao_merge_up_flag)		
sao_offset_cabac(rx, ry, cIdx)		
}		
}		
}		

FIG. 28B

		Descriptor
sao_offset_cabac(rx, ry, cIdx) {		
sao_type_idx [cIdx][rx][ry]	Index number Pixel classifying method	ae(v)
if(sao_type_idx[cIdx][rx][ry] != 0) {		
for(i = 0; i < 4; i++)		
sao_offset [cIdx][rx][ry][i]		ae(v)
}		
if(sao_type_idx[cIdx][rx][ry] == 5) {		
for(i = 0; i < 4; i++) {		
if(sao_offset[cIdx][rx][ry] != 0)		
sao_offset_sign [cIdx][rx][ry][i]		ae(v)
}		
sao_band_position [cIdx][rx][ry]		ae(v)
}		
}		

FIG. 28C

Syntax element	ctxIdxTable, ctxIdxOffset	binIdx				
		0	1	2	3	>=4
sao_on_flag	Table 9-5'	0	cIdx	na	na	na
		3	cIdx	na	na	na
		6	cIdx	na	na	na
sao_merge_left_flag	Table 9-5	0	cIdx	na	na	na
		3	cIdx	na	na	na
		6	cIdx	na	na	na
sao_merge_up_flag	Table 9-6	0	0	na	na	na
		1	0	na	na	na
		2	0	na	na	na
sao_type_idx	Table 9-7	0	0	1	1	1
		2	0	1	1	1
		4	0	1	1	1
sao_offset	Table 9-8	0	0	1	1	1
		2	0	1	1	1
		4	0	1	1	1
sao_offset_sign	na	na	na(uses Decode Bypass)	na	na	na
		na	na(uses Decode Bypass)	na	na	na
		na	na(uses Decode Bypass)	na	na	na
sao_band_position	na	na	na(uses Decode Bypass)	na	na	na
		na	na(uses Decode Bypass)	na	na	na
		na	na(uses Decode Bypass)	na	na	na

FIG. 28D

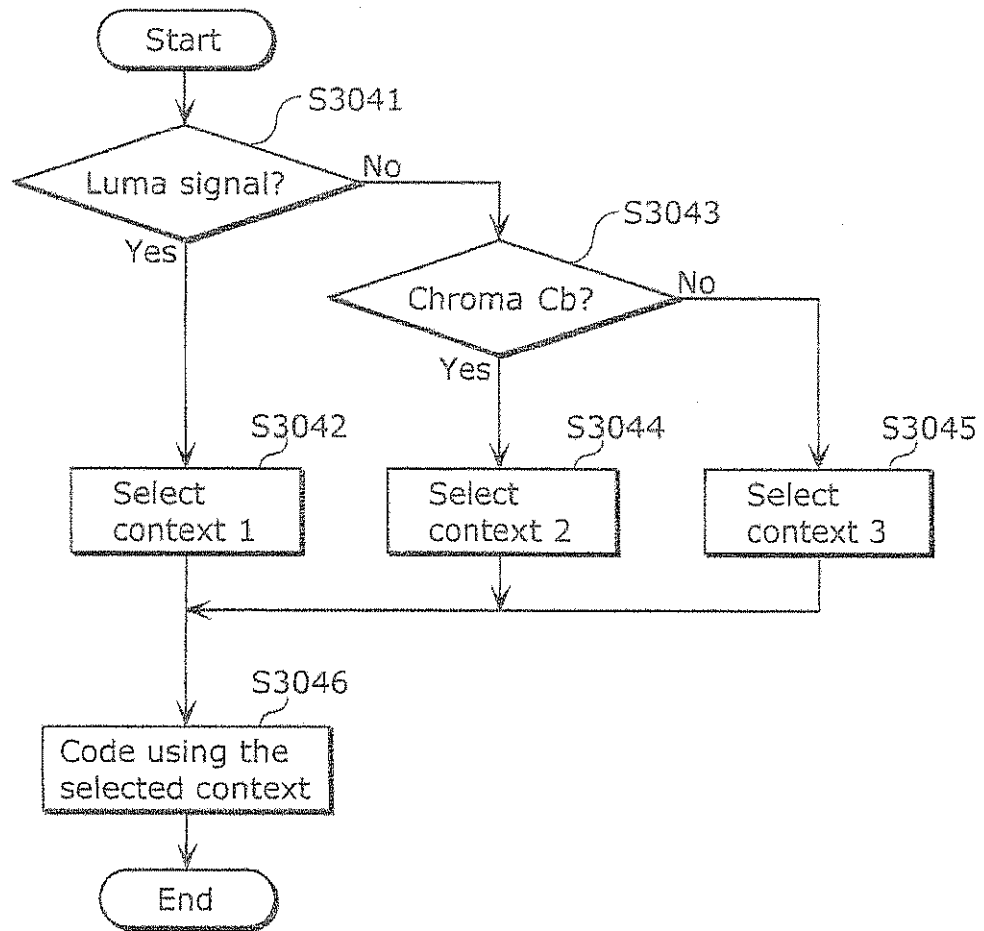


FIG. 28E

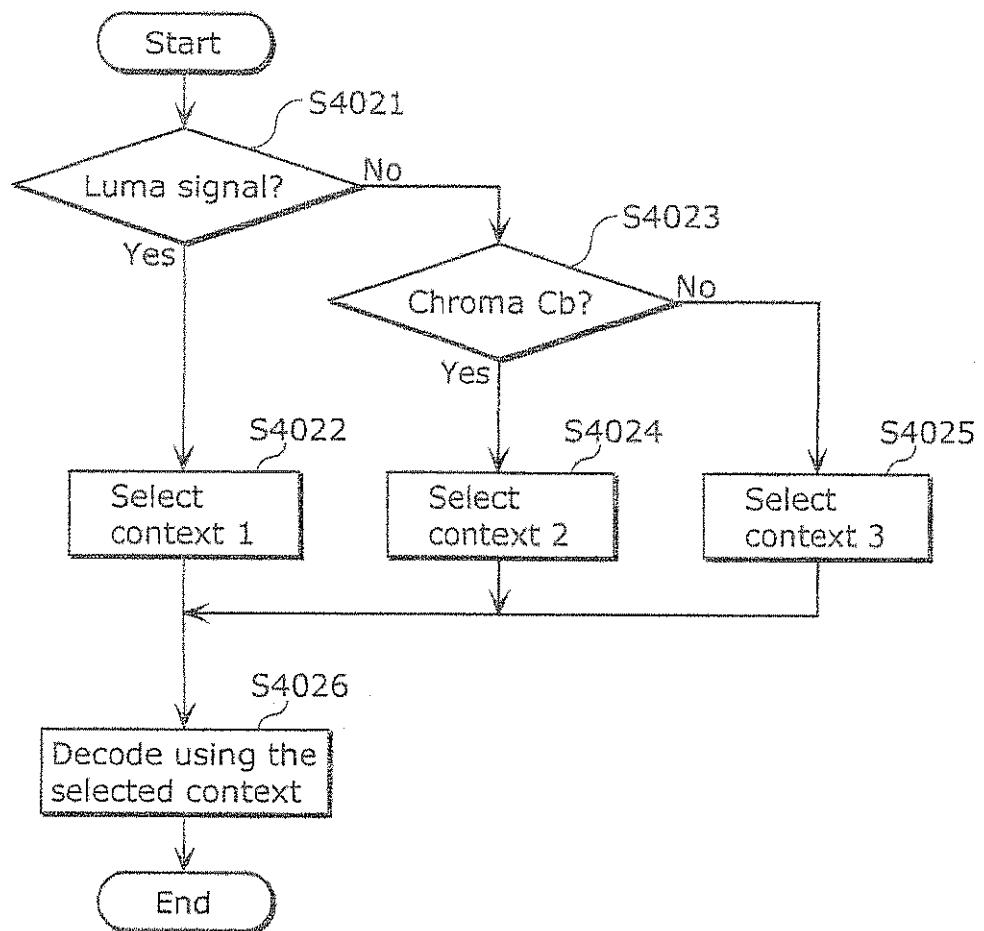


FIG. 29

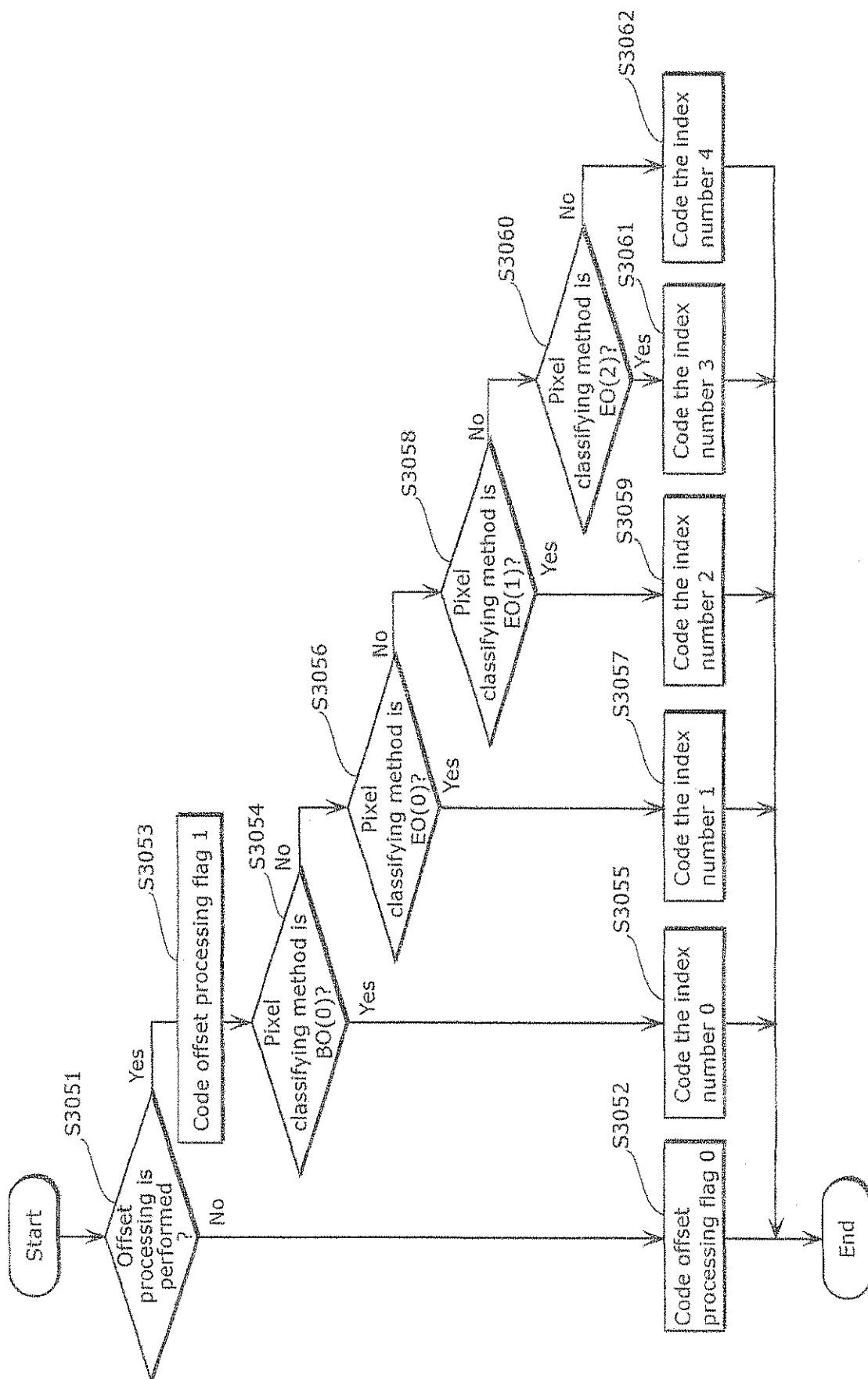


FIG. 30

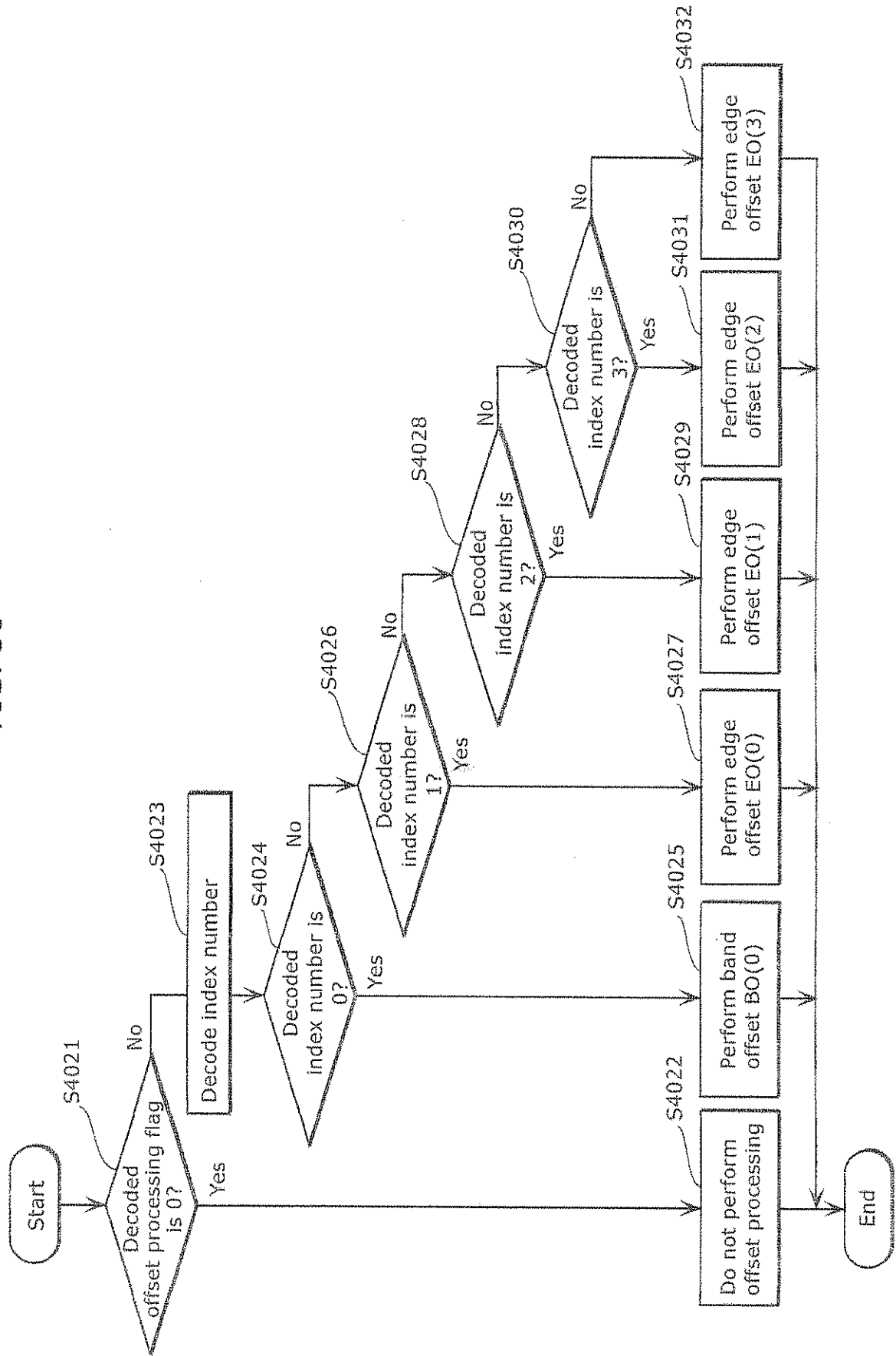


FIG. 31

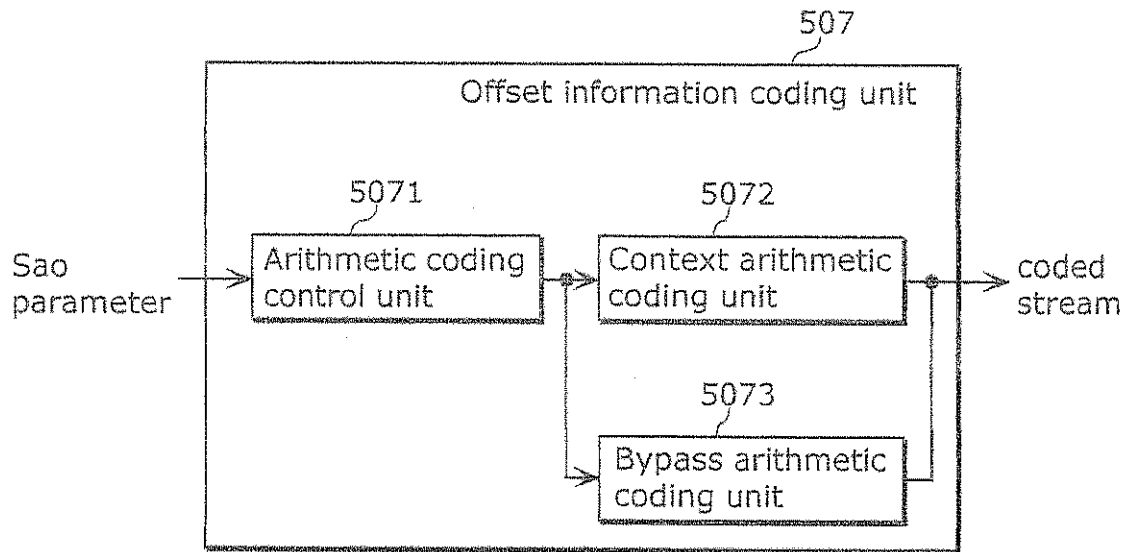


FIG. 32

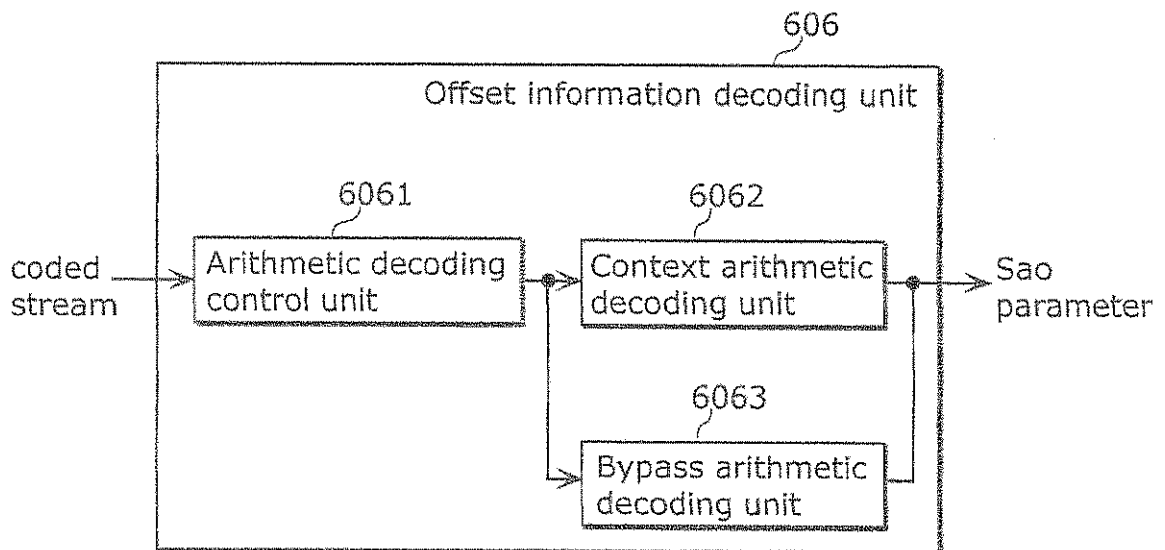


FIG. 33

Syntax element	ctxIdxTable, ctxIdxOffset		binIdx				
			0	1	2	3	>=4
sao_merge_left_flag	na	na	na(uses De- code Bypass)	na	na	na	na
		na	na(uses De- code Bypass)	na	na	na	na
		na	na(uses De- code Bypass)	na	na	na	na
sao_merge_up_flag	na	na	na(uses De- code Bypass)	na	na	na	na
		na	na(uses De- code Bypass)	na	na	na	na
		na	na(uses De- code Bypass)	na	na	na	na
sao_type_idx	Table 9-X	0	0	na(uses De- code Bypass)	na	na	na
		1	0	na(uses De- code Bypass)	na	na	na
		2	0	na(uses De- code Bypass)	na	na	na
sao_offset	na	na	na(uses De- code Bypass)	na	na	na	na
		na	na(uses De- code Bypass)	na	na	na	na
		na	na(uses De- code Bypass)	na	na	na	na
sao_offset_sign	na	na	na(uses De- code Bypass)	na	na	na	na
		na	na(uses De- code Bypass)	na	na	na	na
		na	na(uses De- code Bypass)	na	na	na	na
sao_band_position	na	na	na(uses De- code Bypass)	na	na	na	na
		na	na(uses De- code Bypass)	na	na	na	na
		na	na(uses De- code Bypass)	na	na	na	na

FIG. 34

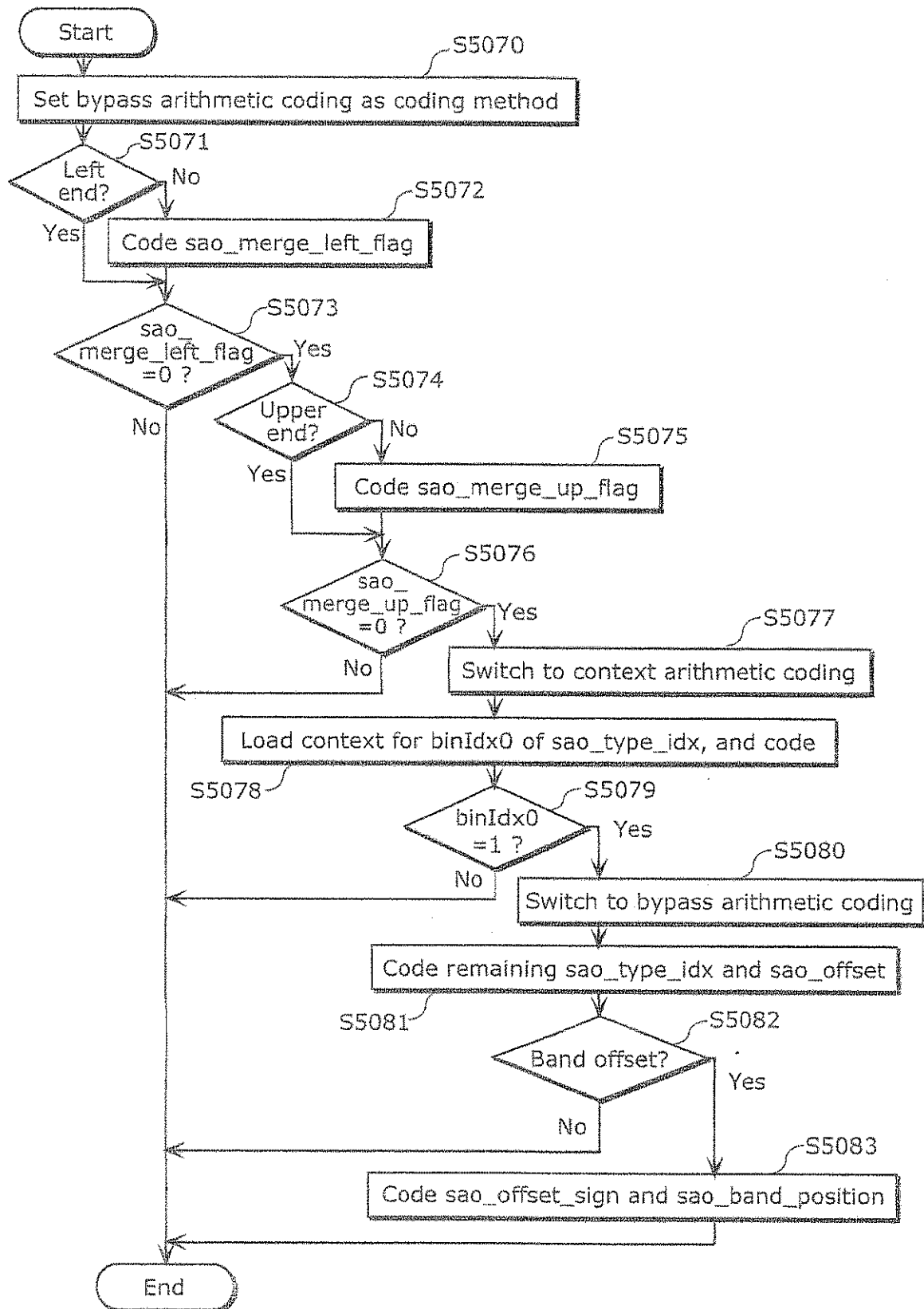


FIG. 35

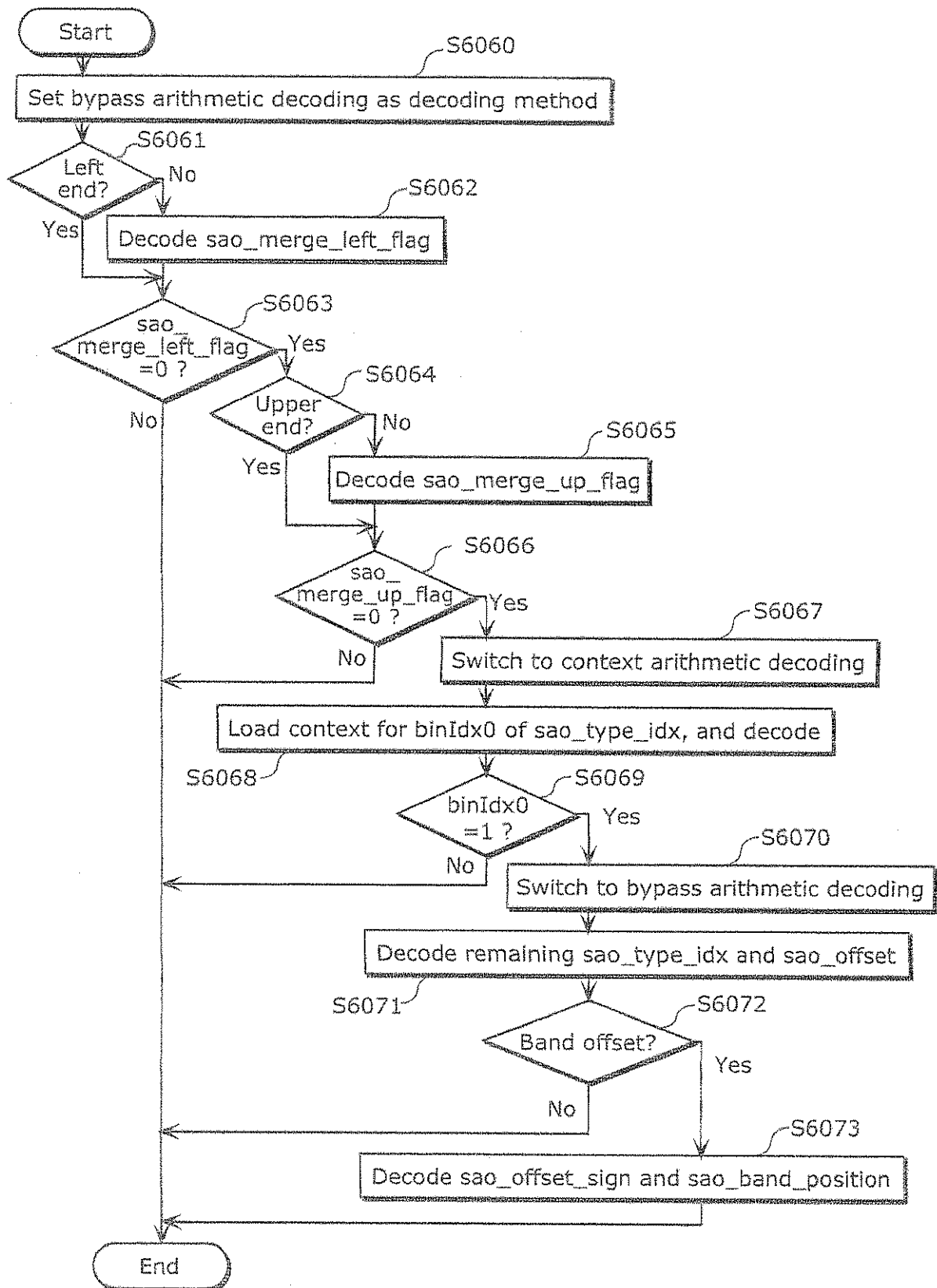


FIG. 36A

Syntax element	ctxIdxTable, ctxIdxOffset		binIdx				
			0	1	2	3	>=4
sao_on_flag	Table 9-X	0	0	na	na	na	na
		1	0	na	na	na	na
		2	0	na	na	na	na
sao_merge_left_flag	na	na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
sao_merge_up_flag	na	na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
sao_type_idx	na	na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
sao_offset	na	na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
sao_offset_sign	na	na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
sao_band_position	na	na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na

FIG. 36B

Index	Bit allocation	Offset processing flag
0	0	Do not perform offset processing
1	1	Perform offset processing

Index	Bit allocation	Pixel classifying method
0	0	Edge offset (0)
1	10	Edge offset (1)
2	110	Edge offset (2)
3	1110	Edge offset (3)
4	1111	Band offset

FIG. 36C

Configuration	BD-rate
All Intra Main	0.1%
Random Access Main	-0.1%
Low Delay B Main	0.0%
All Intra HE10	0.1%
Random Access HE10	0.0%
Low delay B HE10	0.0%

FIG. 37

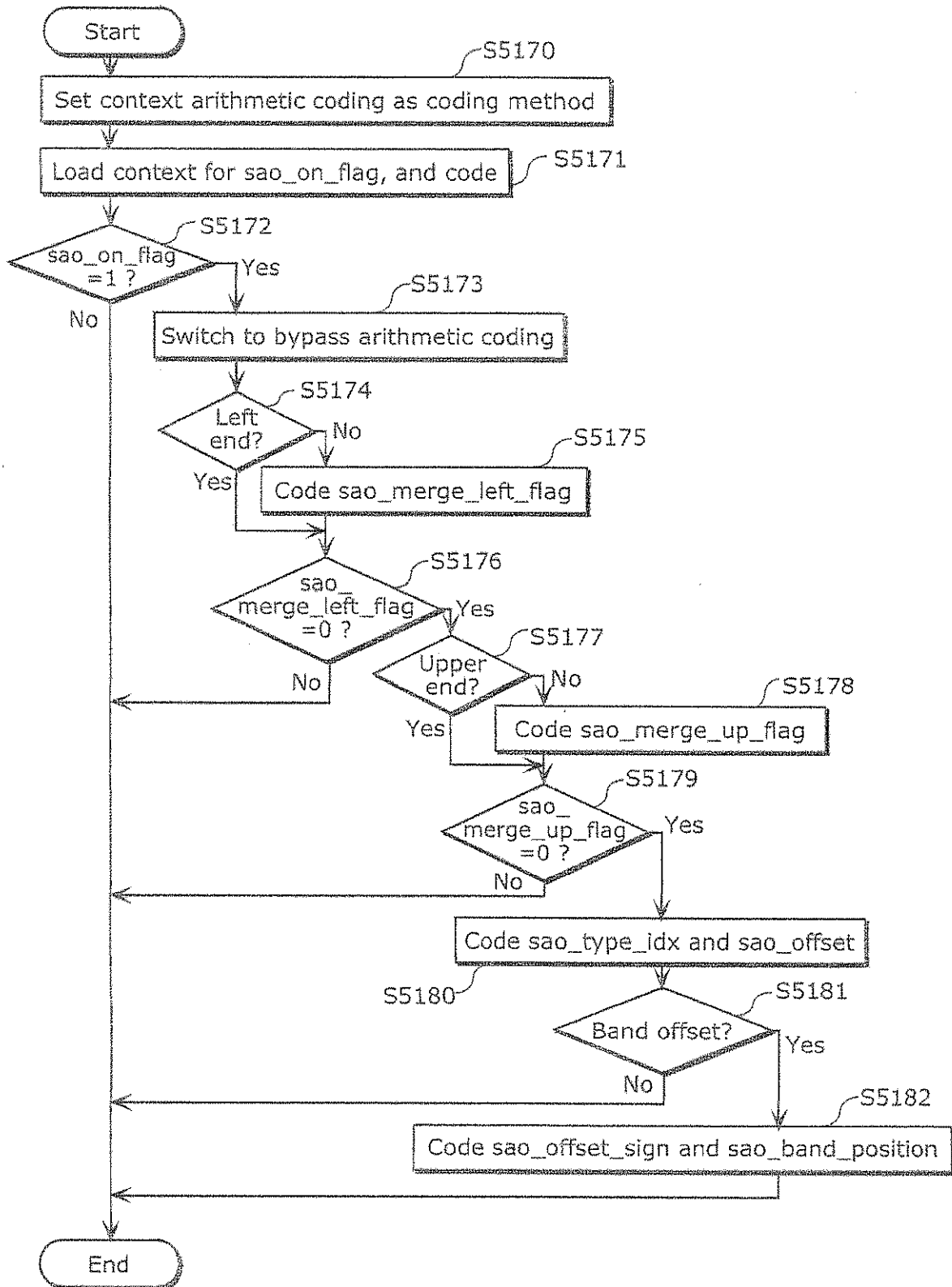


FIG. 38

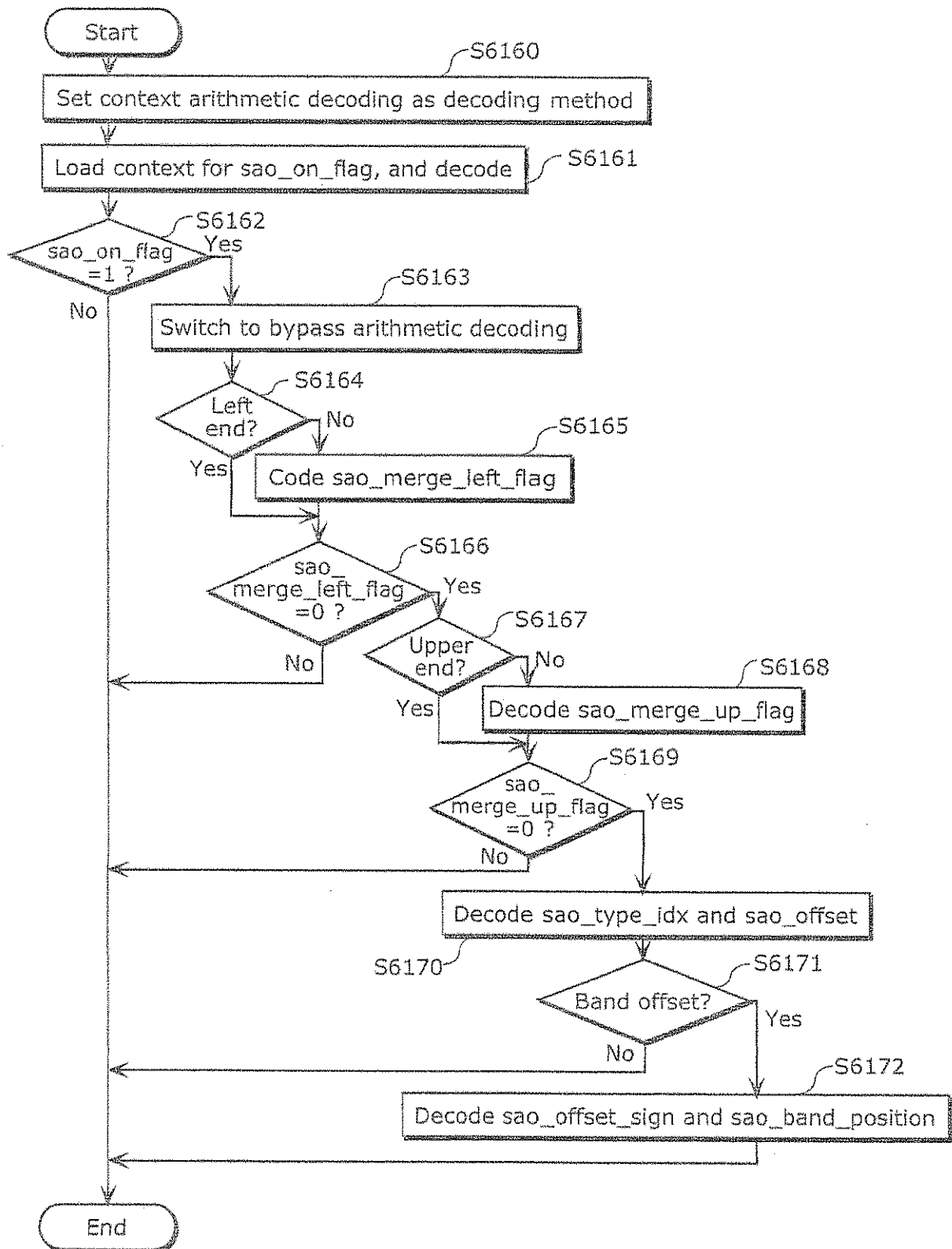


FIG. 39A

Syntax element	ctxIdxTable, ctxIdxOffset		binIdx				
			0	1	2	3	>=4
sao_on_flag	Table 9-X	0	0	na	na	na	na
		1	0	na	na	na	na
		2	0	na	na	na	na
sao_merge_left_flag	Table 9-XX	0	0	na	na	na	na
		1	0	na	na	na	na
		2	0	na	na	na	na
sao_merge_up_flag	Table 9-XXX	0	0	na	na	na	na
		1	0	na	na	na	na
		2	0	na	na	na	na
sao_type_idx	na	na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
sao_offset	na	na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
sao_offset_sign	na	na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
sao_band_position	na	na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na
		na	na(uses Decode Bypass)	na	na	na	na

FIG. 39B

Configuration	BD-rate
All Intra Main	0.1 %
Random Access Main	0.0 %
Low Delay B Main	0.0 %
All Intra HE10	0.1 %
Random Access HE10	-0.1 %
Low delay B HE10	0.0 %

FIG. 40

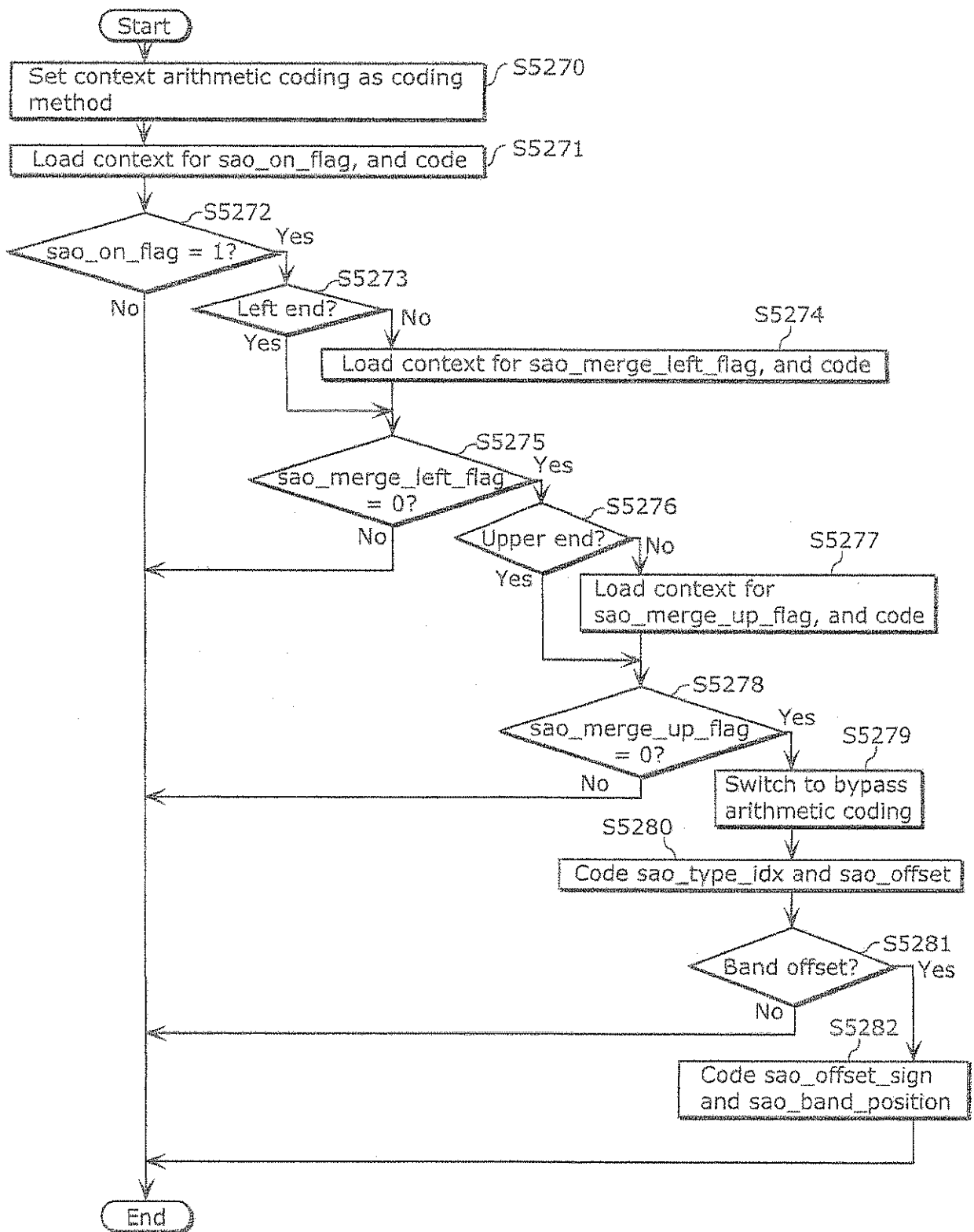


FIG. 41

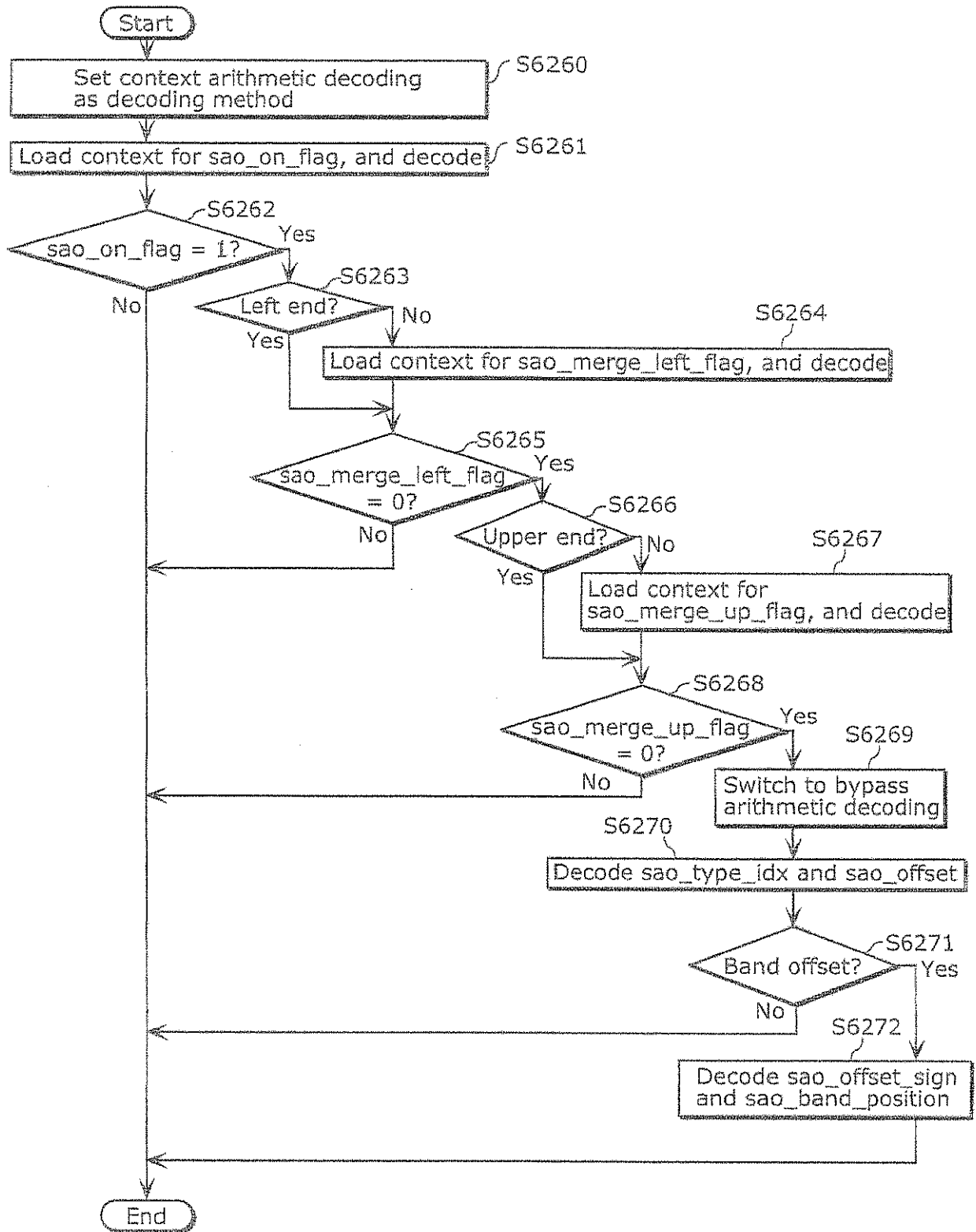


FIG. 42

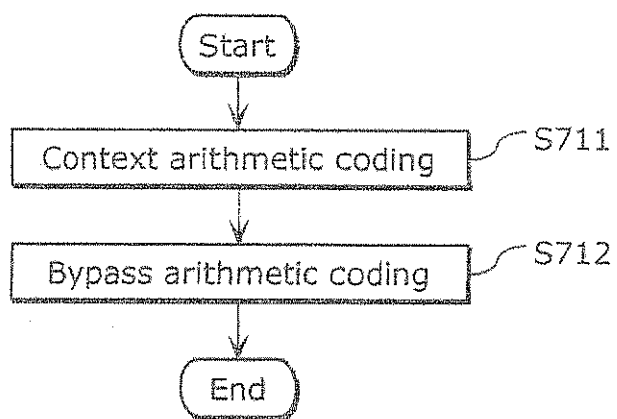


FIG. 43

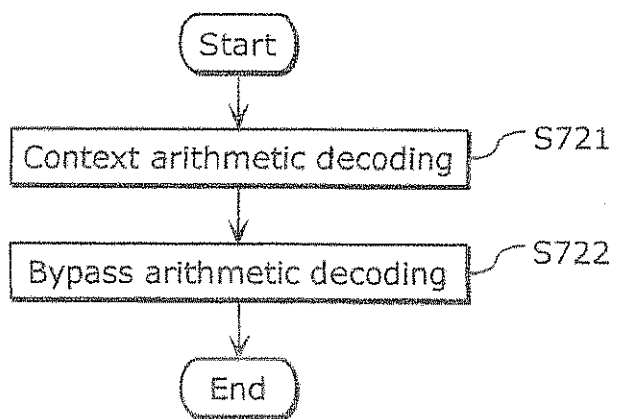


FIG. 44

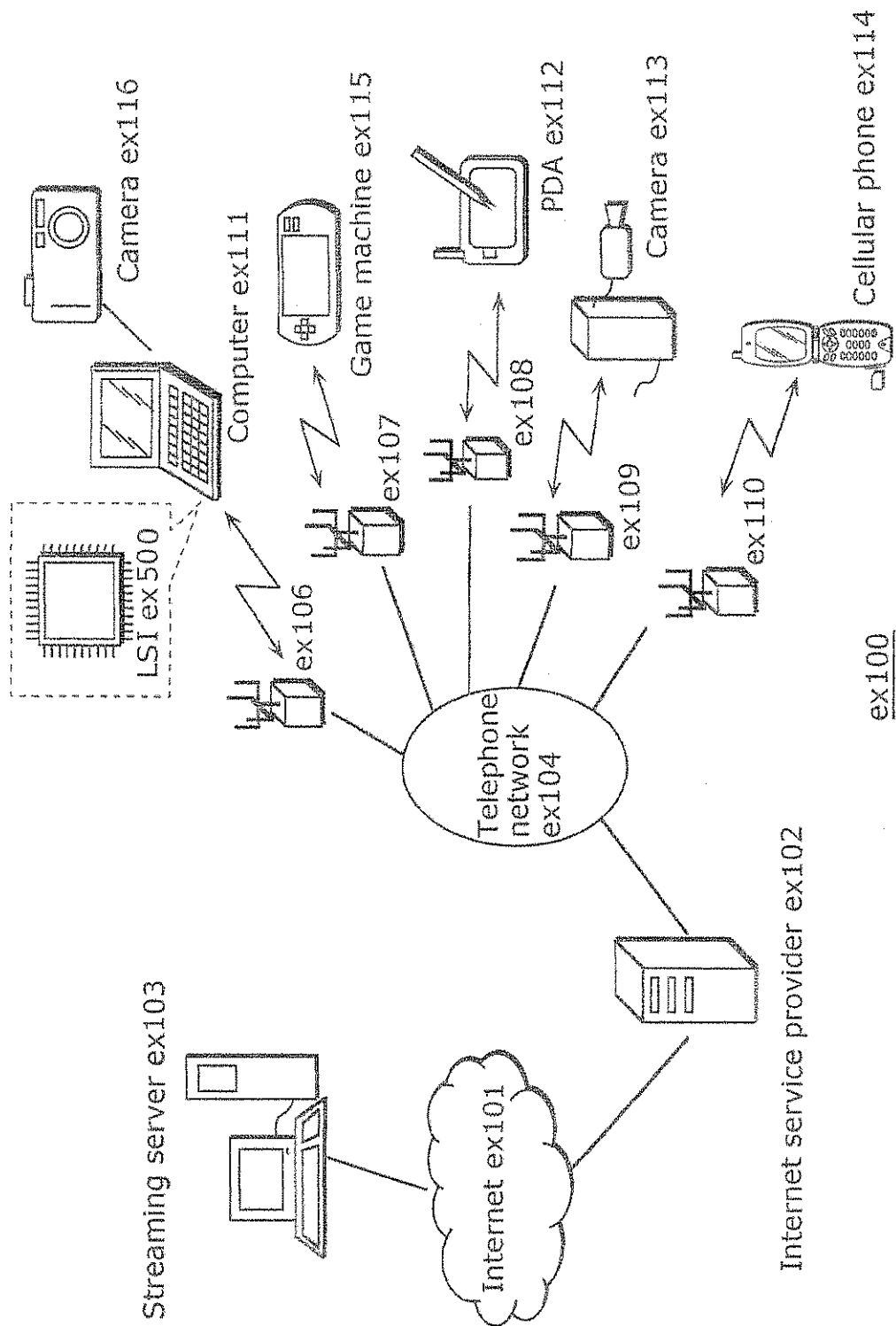


FIG. 45

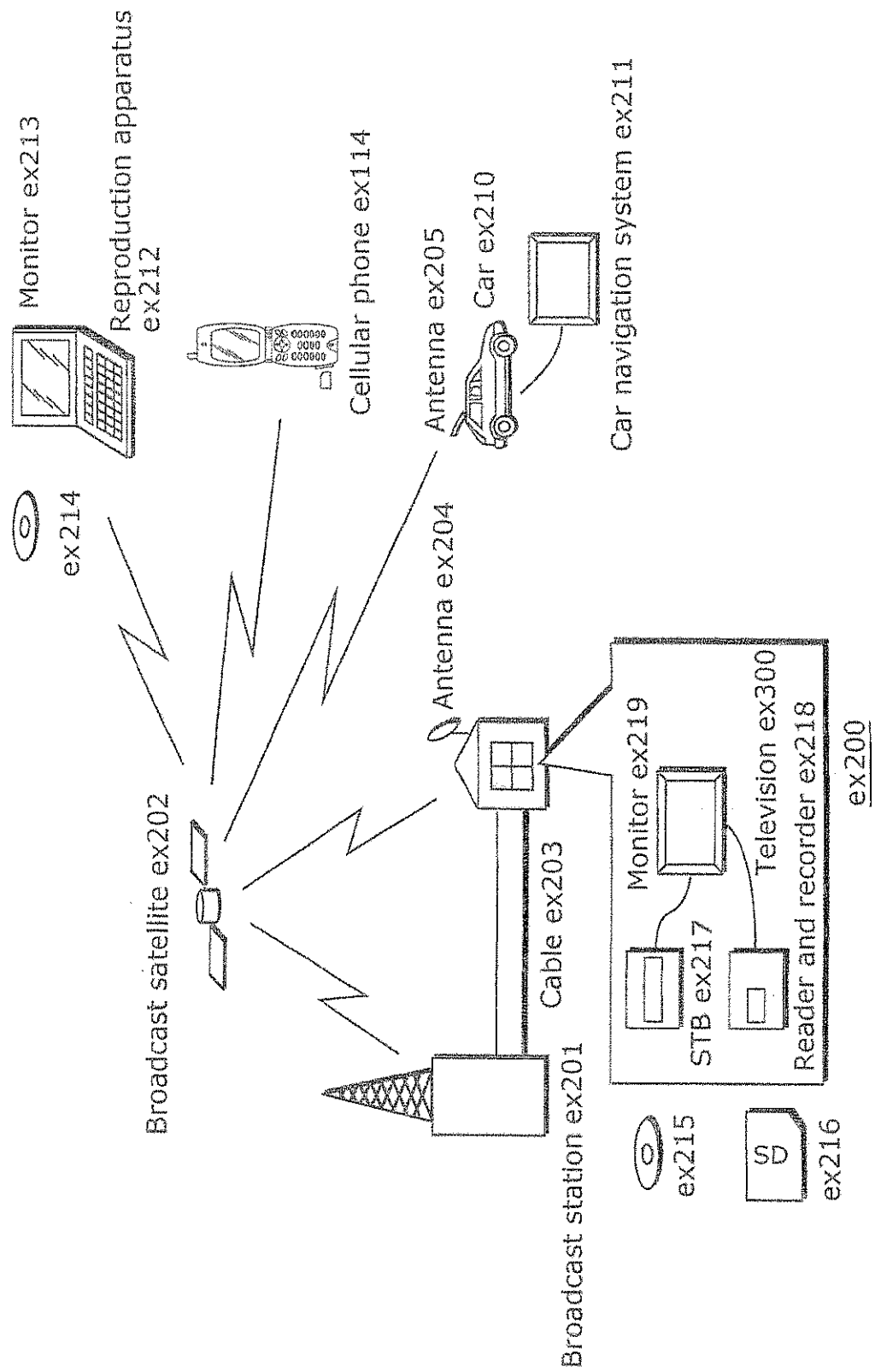


FIG. 46

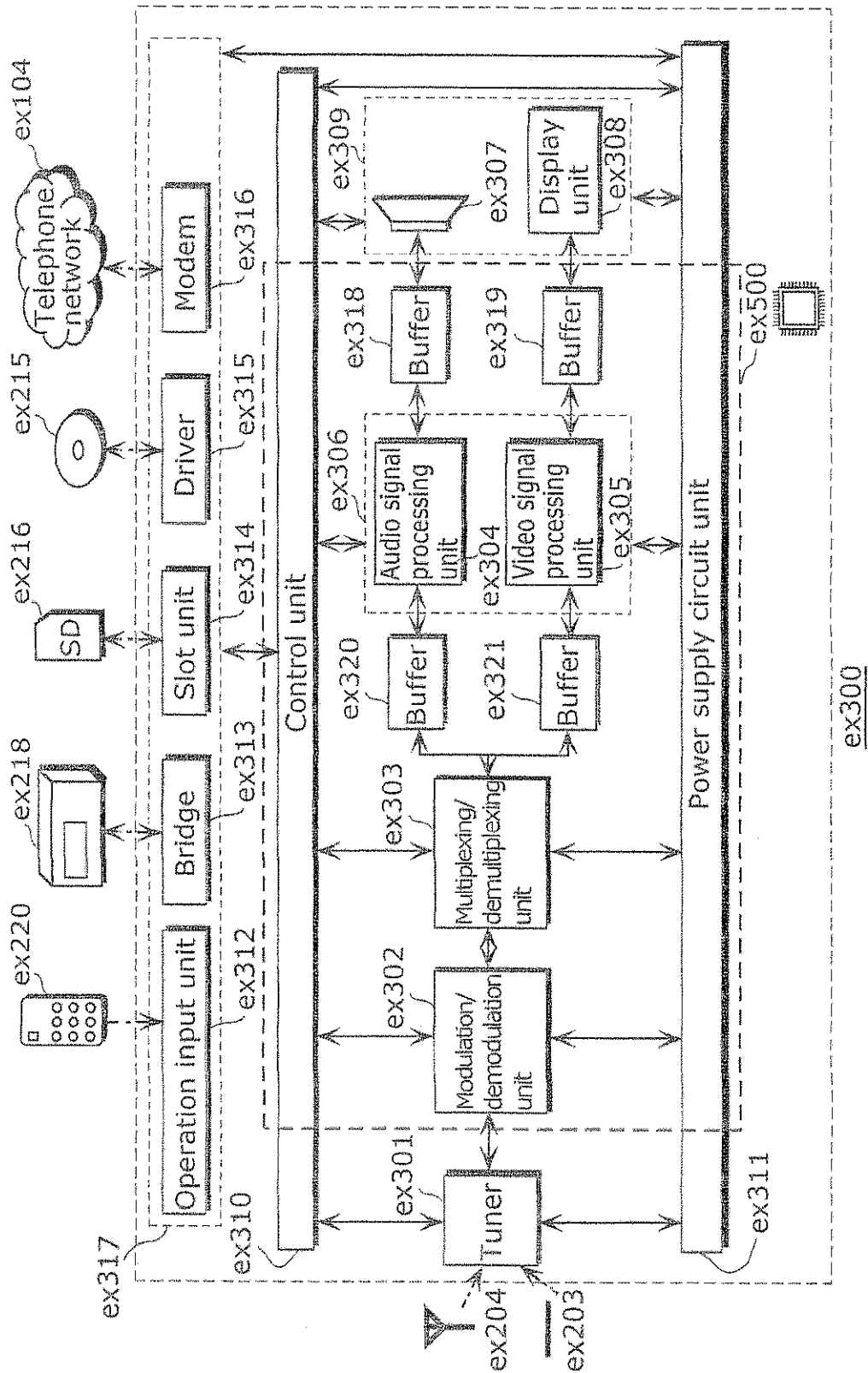
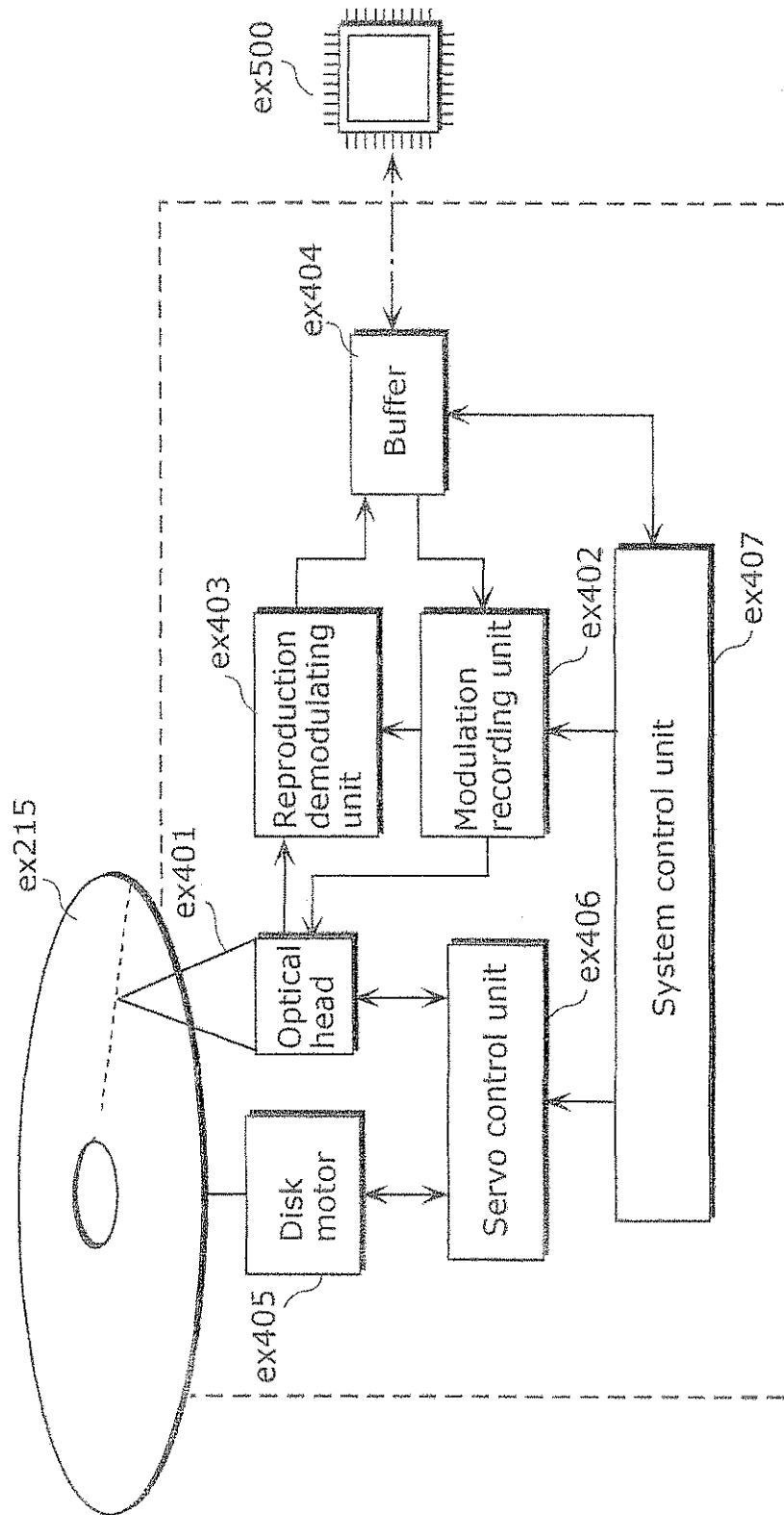


FIG. 47



ex400

FIG. 48

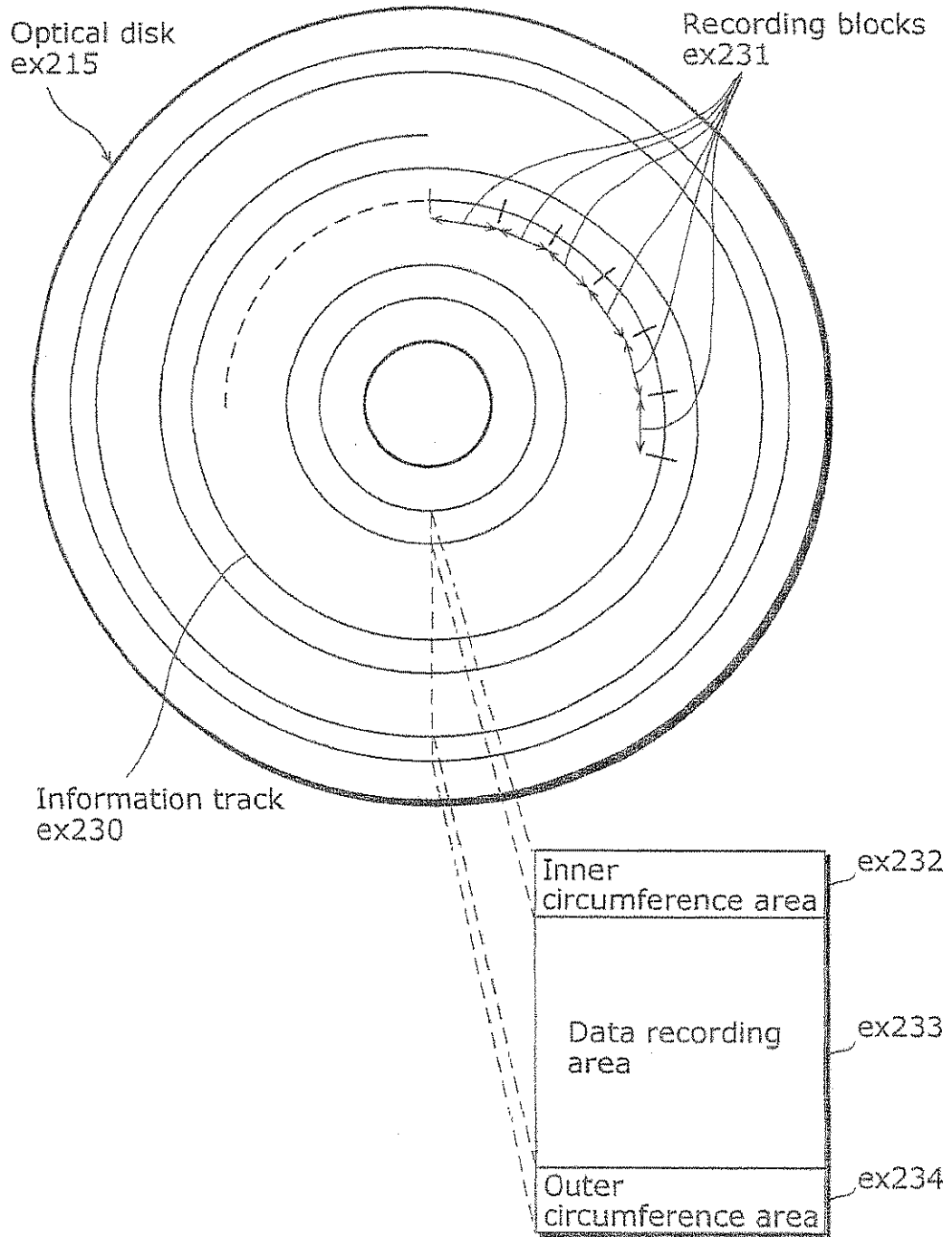


FIG. 49A

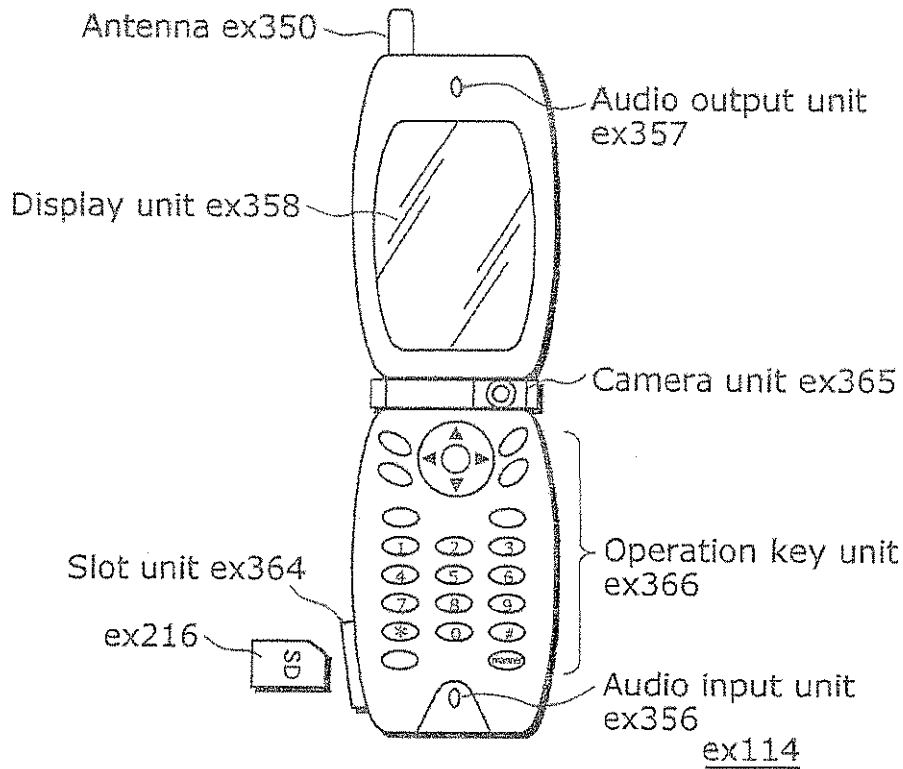


FIG. 49B

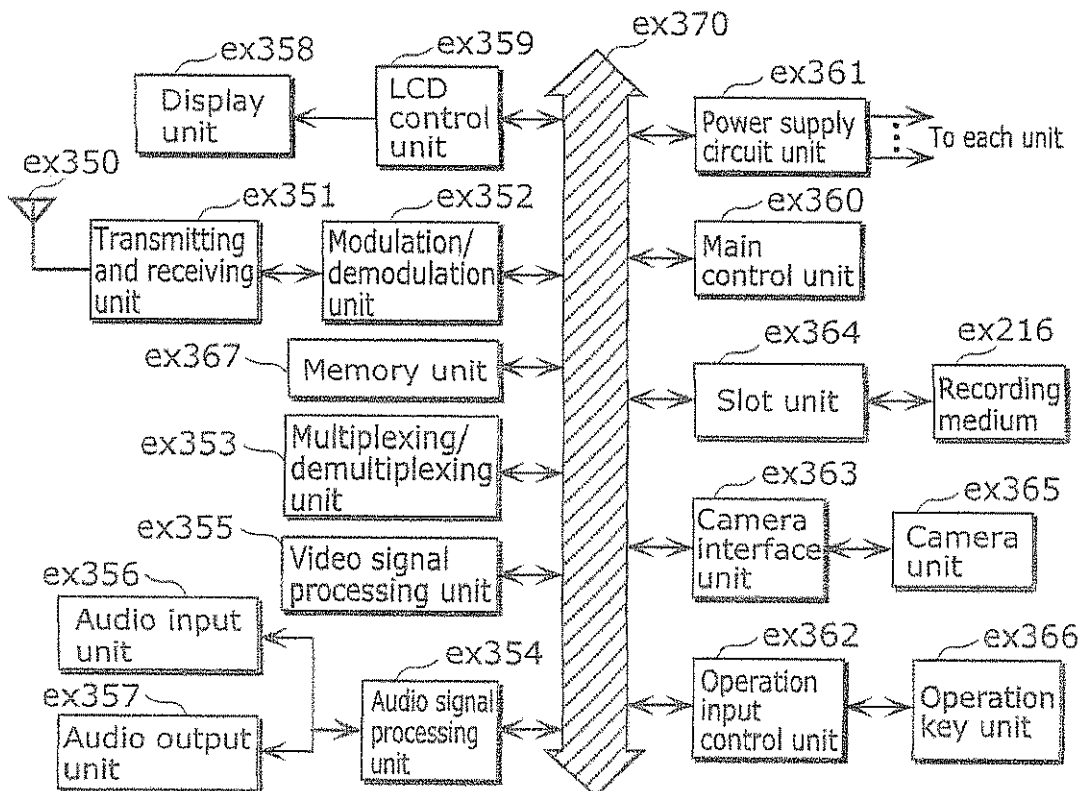


FIG. 50

Video stream (PID=0x1011, Primary video)
Audio stream (PID=0x1100)
Audio stream (PID=0x1101)
Presentation graphics stream (PID=0x1200)
Presentation graphics stream (PID=0x1201)
Interactive graphics stream (PID=0x1400)
Video stream (PID=0x1B00, Secondary video)
Video stream (PID=0x1B01, Secondary video)

FIG. 51

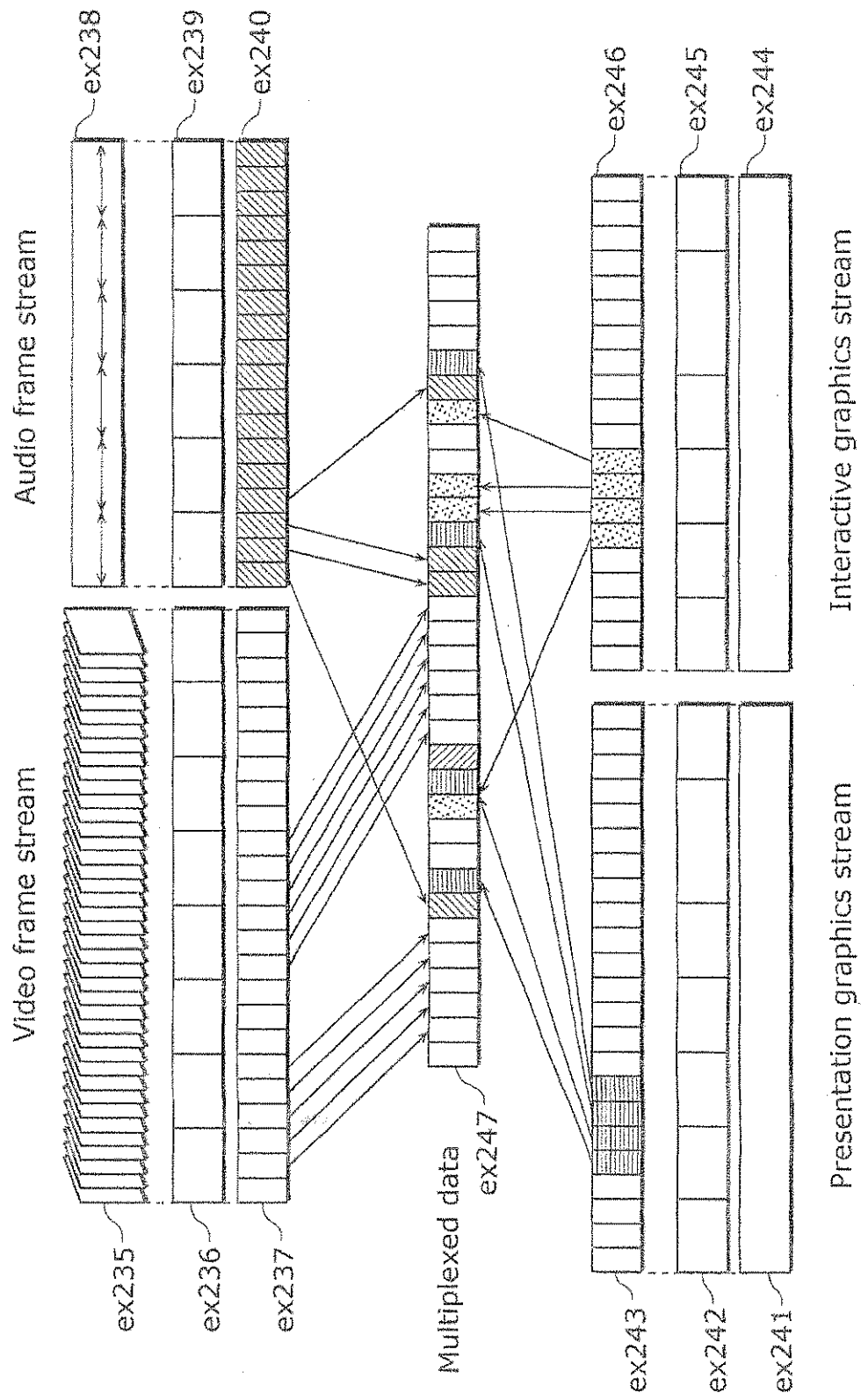


FIG. 52

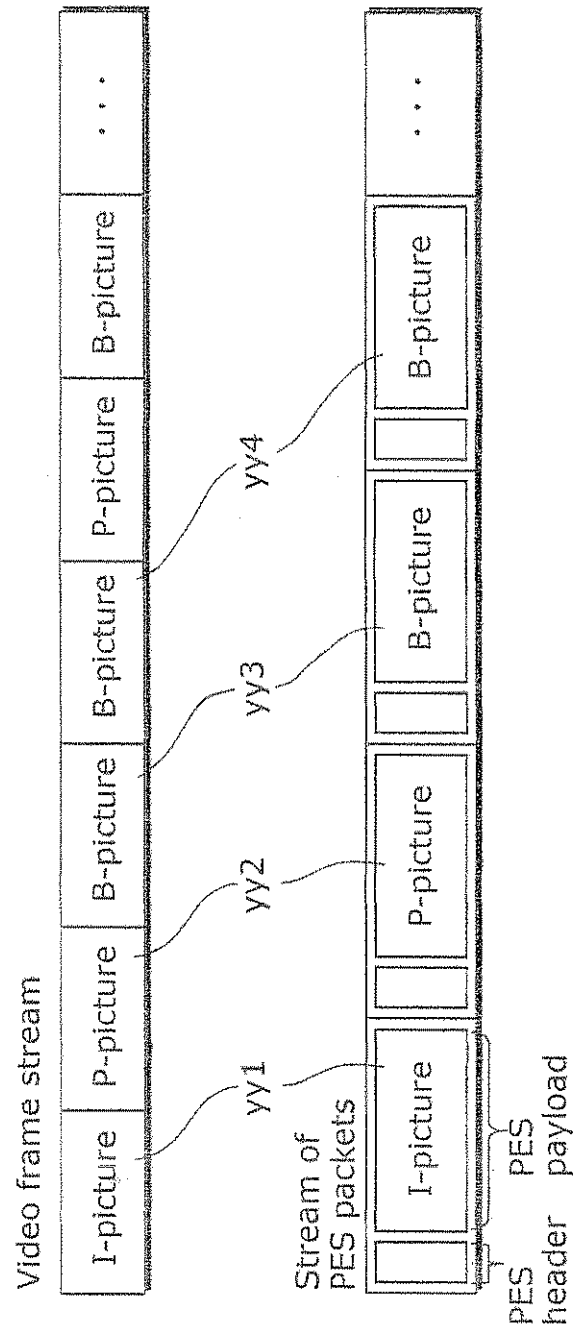
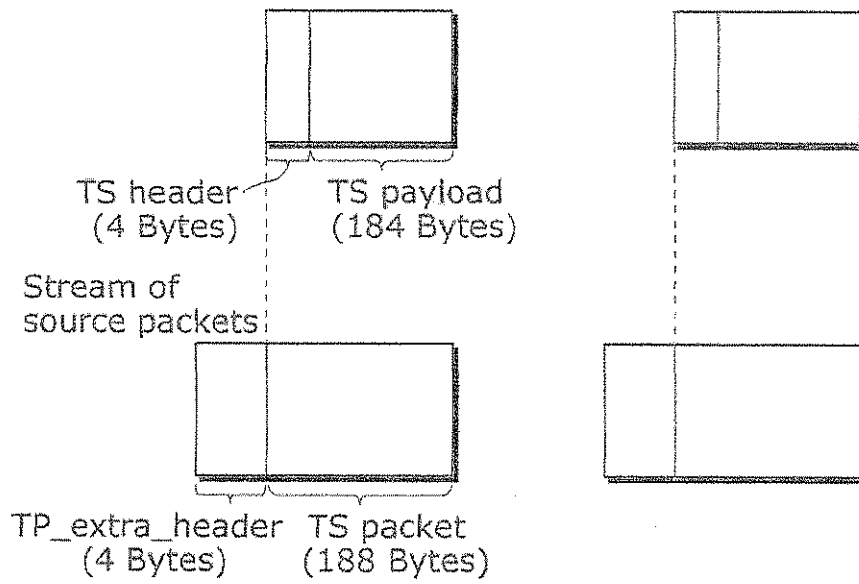


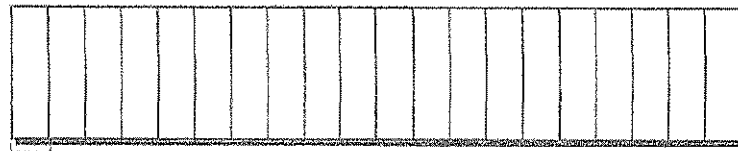
FIG. 53

Stream of TS packets



Multiplexed data

SPN 0 1 2 3 4 5 6 7 ...



Source packet

FIG. 54

Data structure of PMT

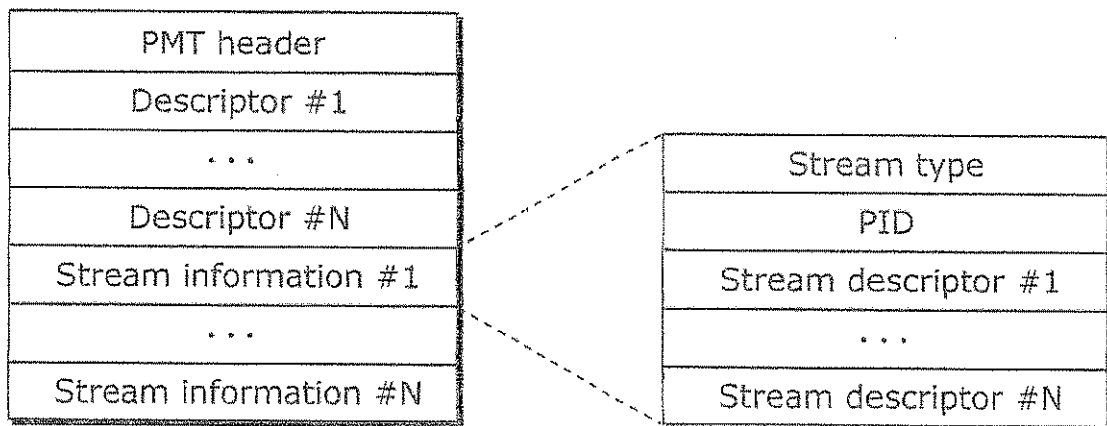


FIG. 55

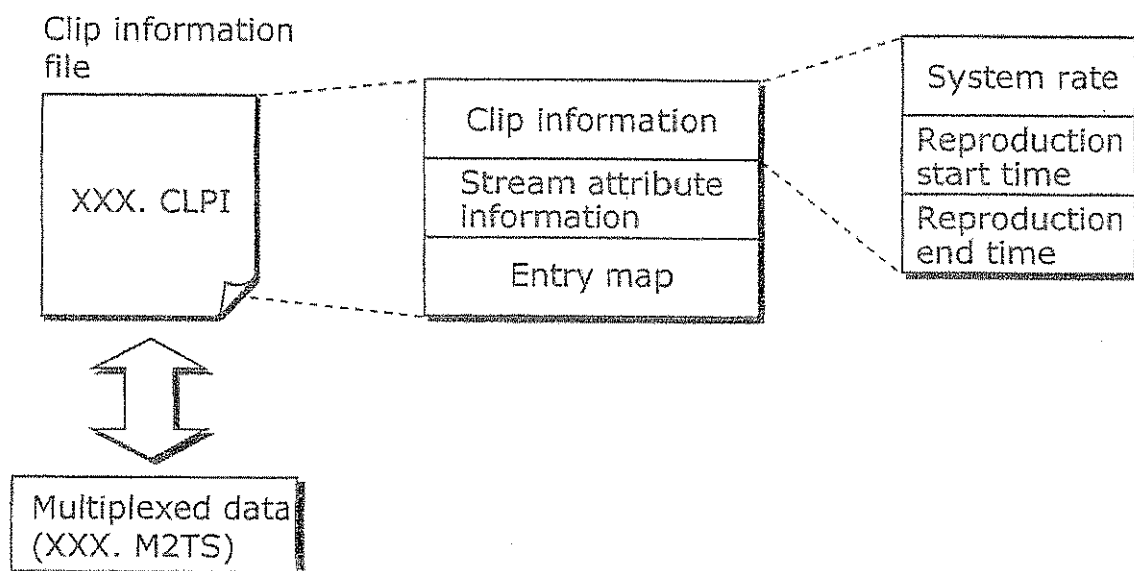


FIG. 56

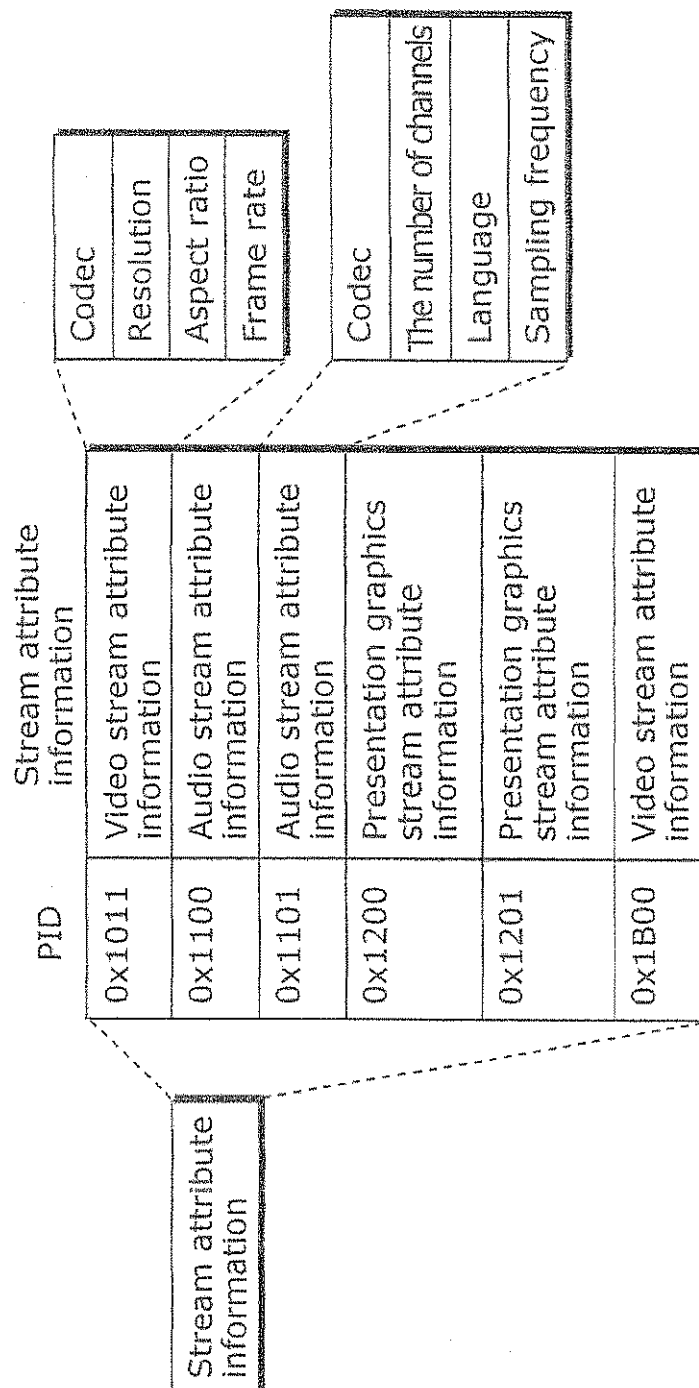


FIG. 57

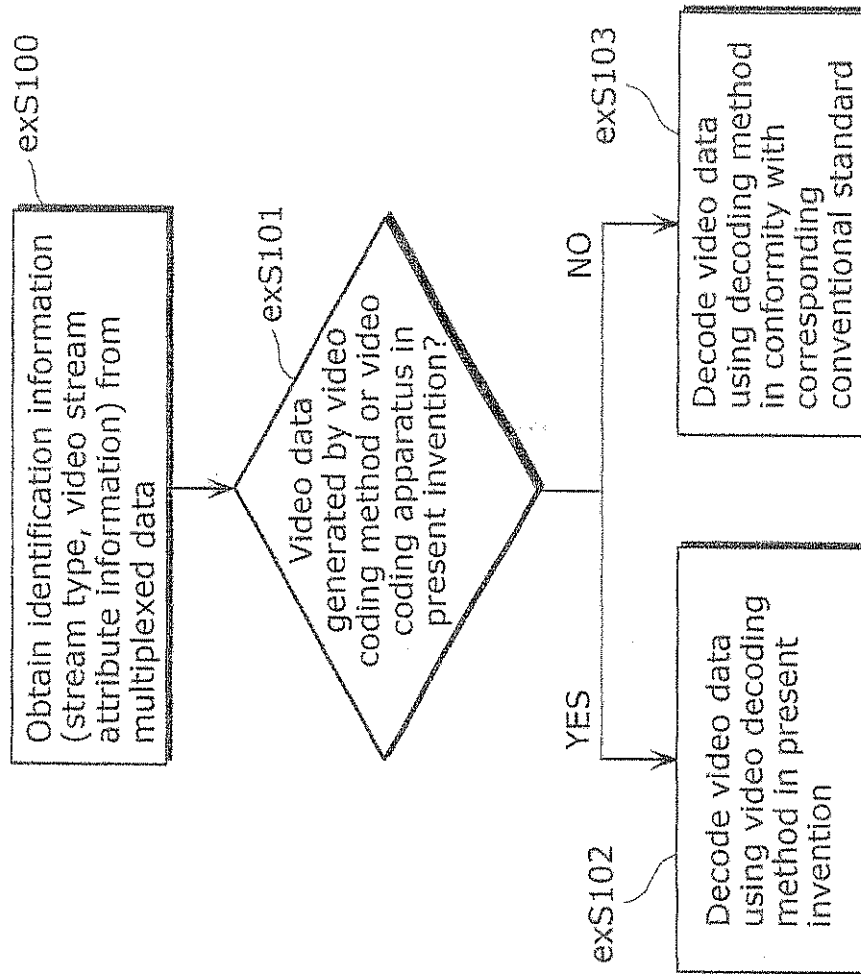


FIG. 58

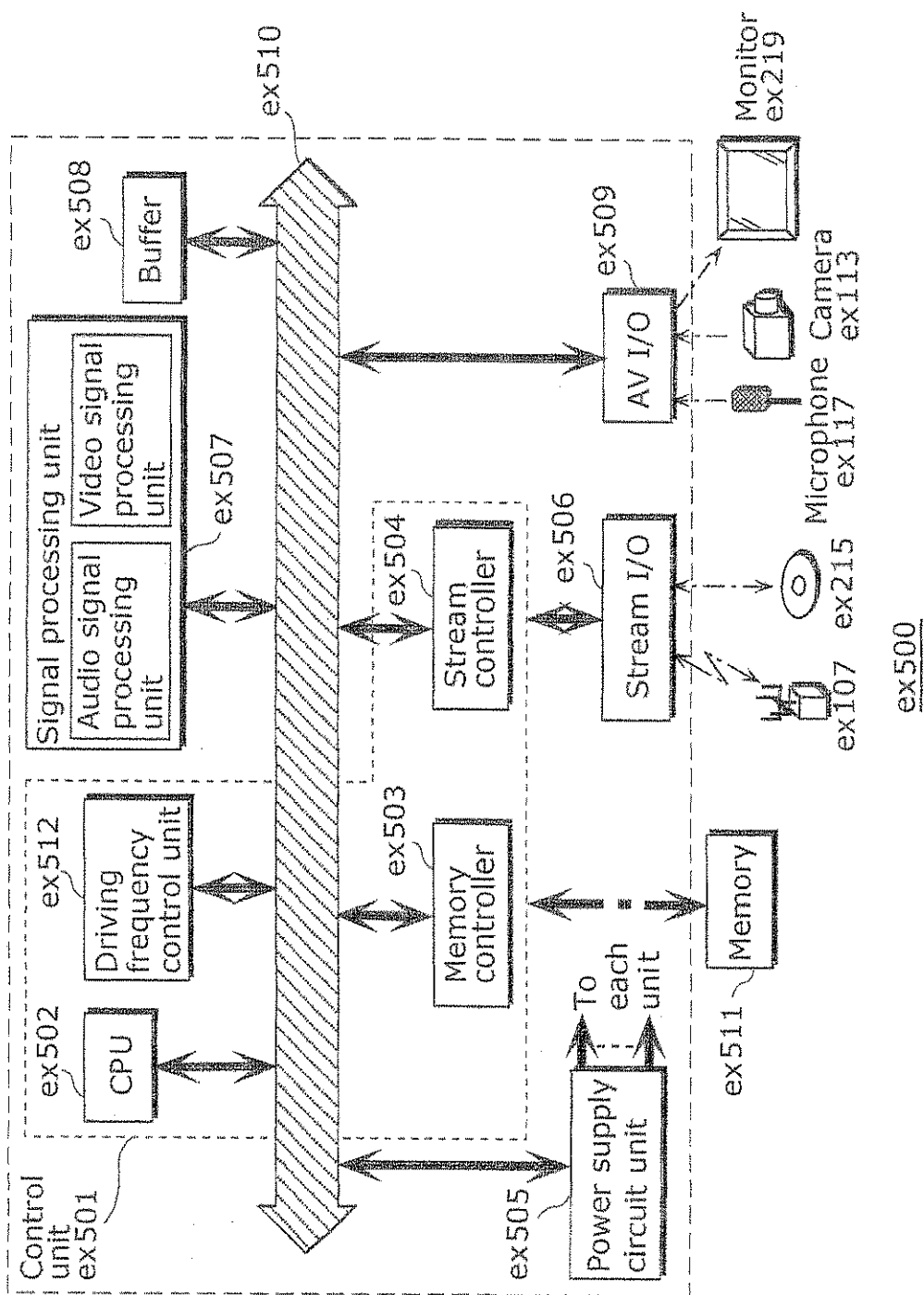


FIG. 59

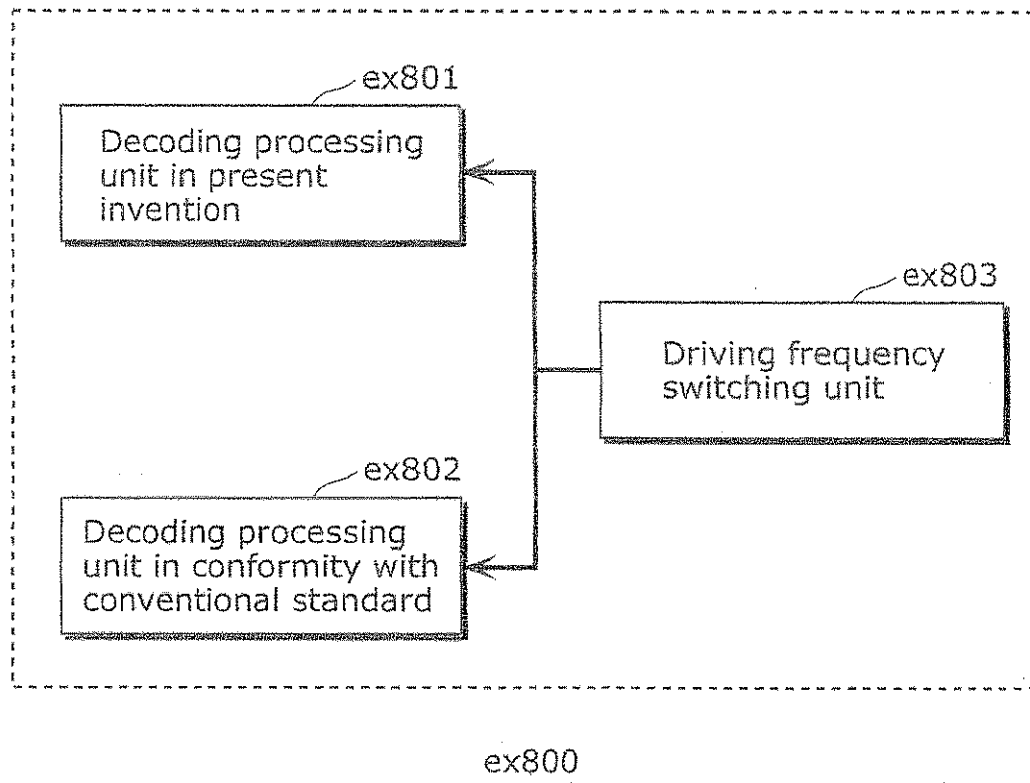


FIG. 60

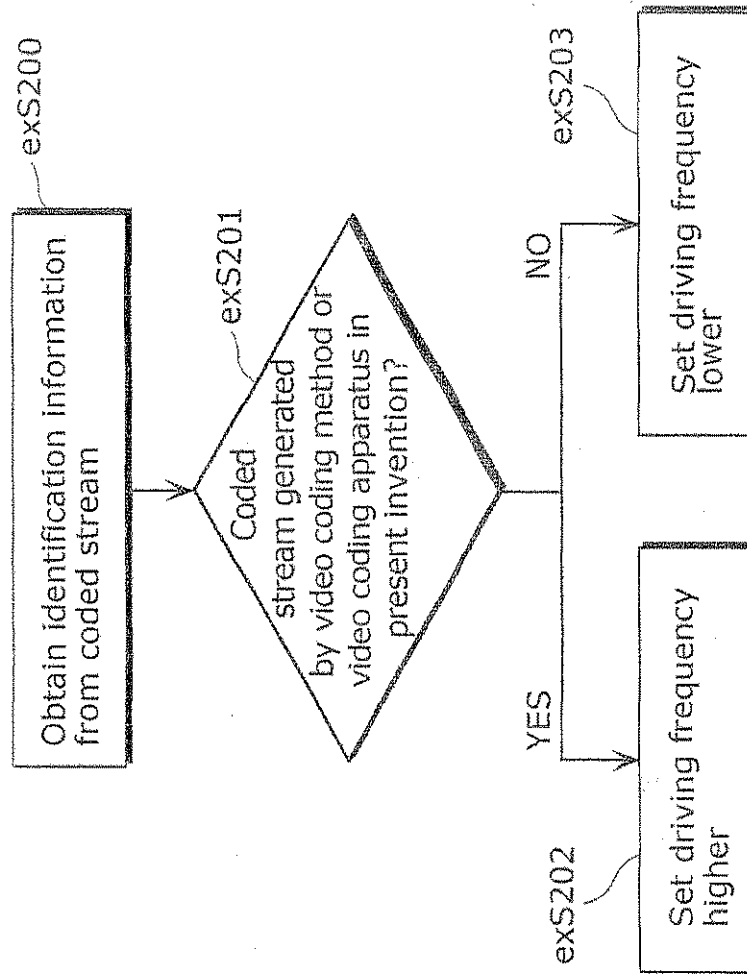


FIG. 61

Corresponding standard	Driving frequency
MPEG-4 AVC	500 MHz
MPEG-2	350 MHz
⋮	⋮

FIG. 62A

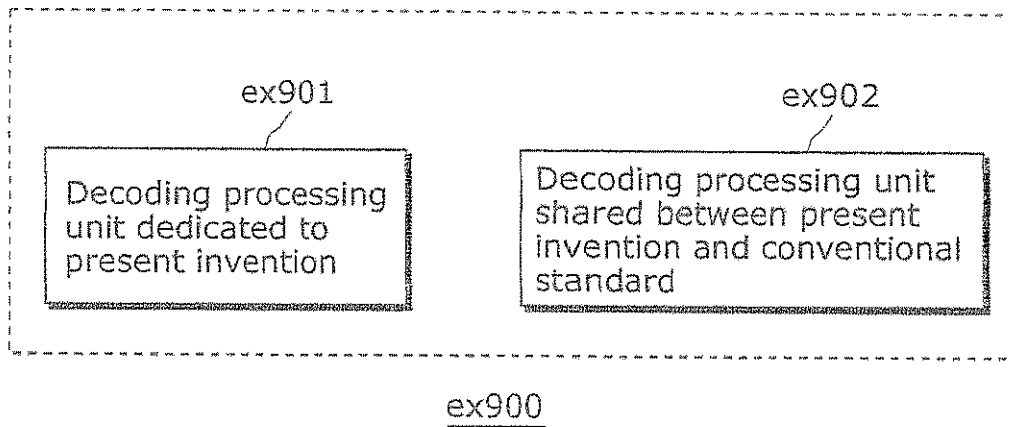


FIG. 62B

