

FIG. 1

1	SET TRANSFORM SKIP FLAG FOR EACH VALUE OF COMPONENT IDENTIFIER
1-1	FOR EACH VALUE OF COMPONENT IDENTIFIER, APPLY TS RESIDUAL ENCODING MODE IN CASE OF TRANSFORM SKIP, APPLY NON-TS RESIDUAL ENCODING MODE IN CASE OF NON-TRANSFORM SKIP
1-1-1	APPLY RESIDUAL ENCODING MODE SELECTION FLAG
1-1-2	APPLY FLAG THAT ENABLES SELECTION OF RESIDUAL ENCODING MODE IN SPECIFIC MODE
1-2	FOR EACH VALUE OF COMPONENT IDENTIFIER, CORRECT QUANTIZATION PARAMETER IN CASE OF TRANSFORM SKIP
1-3	IN CASE OF TRANSFORM SKIP, SHARE CONTEXT VARIABLE BETWEEN LUMINANCE COMPONENT AND COLOR COMPONENT CORRESPONDING TO EACH binIdx IN BIN ROW OF EACH SYNTAX
1-4	SWITCH ENCODING-DECODING METHOD OF SIGN CODE ACCORDING TO TRANSFORM SKIP FLAG CORRESPONDING TO COMPONENT IDENTIFIER
1-4-1	PERFORM BYPASS ENCODING IN CASE OF NOT TRANSFORM SKIP
1-4-2	IN CASE OF TRANSFORM SKIP, IF NUMBER OF RESIDUAL CONTEXT ENCODING BINS IS EQUAL TO OR MORE THAN THRESHOLD, CONTEXT ENCODING IS PERFORMED, AND OTHERWISE PERFORM BYPASS ENCODING

FIG. 2

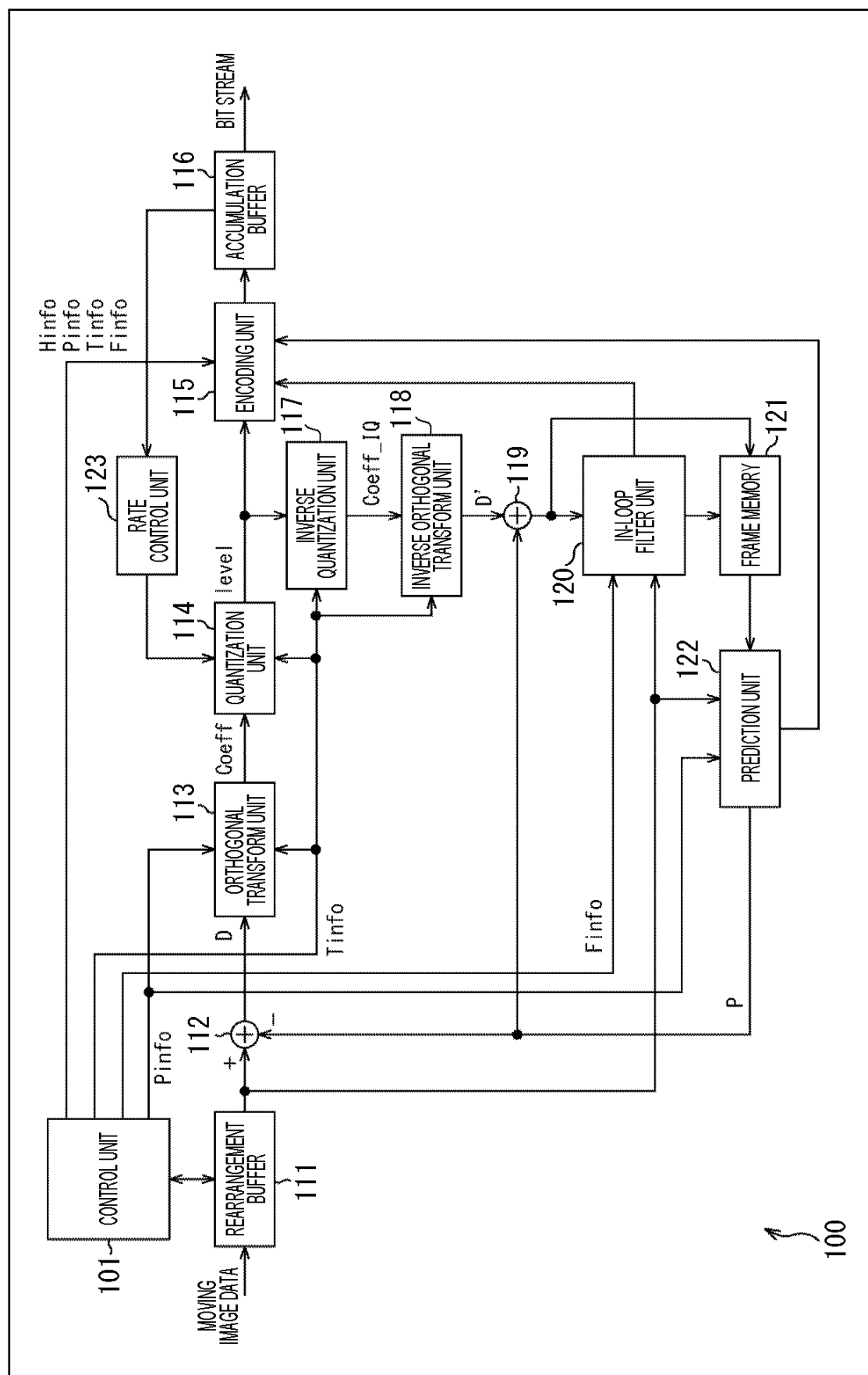


FIG. 3

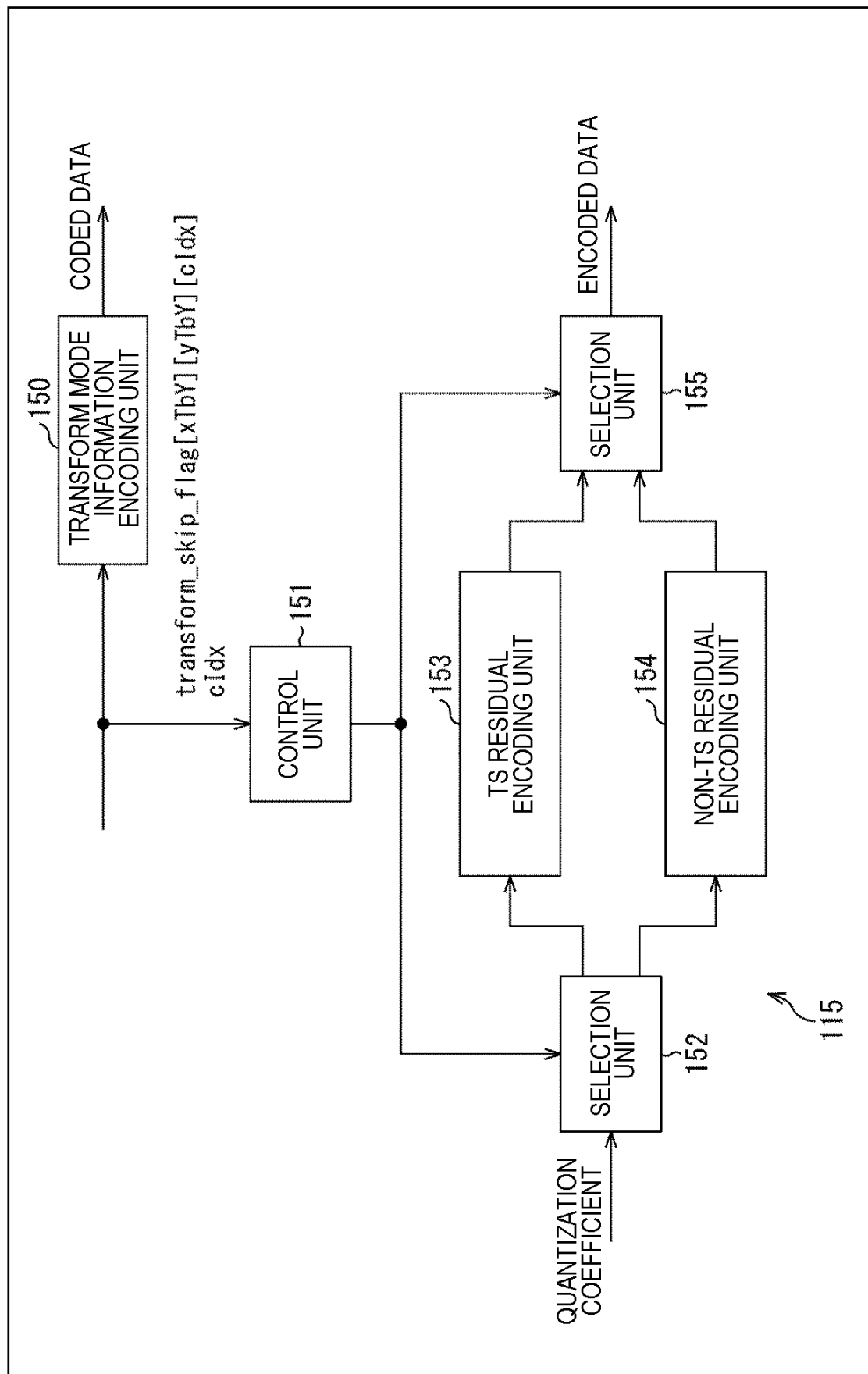
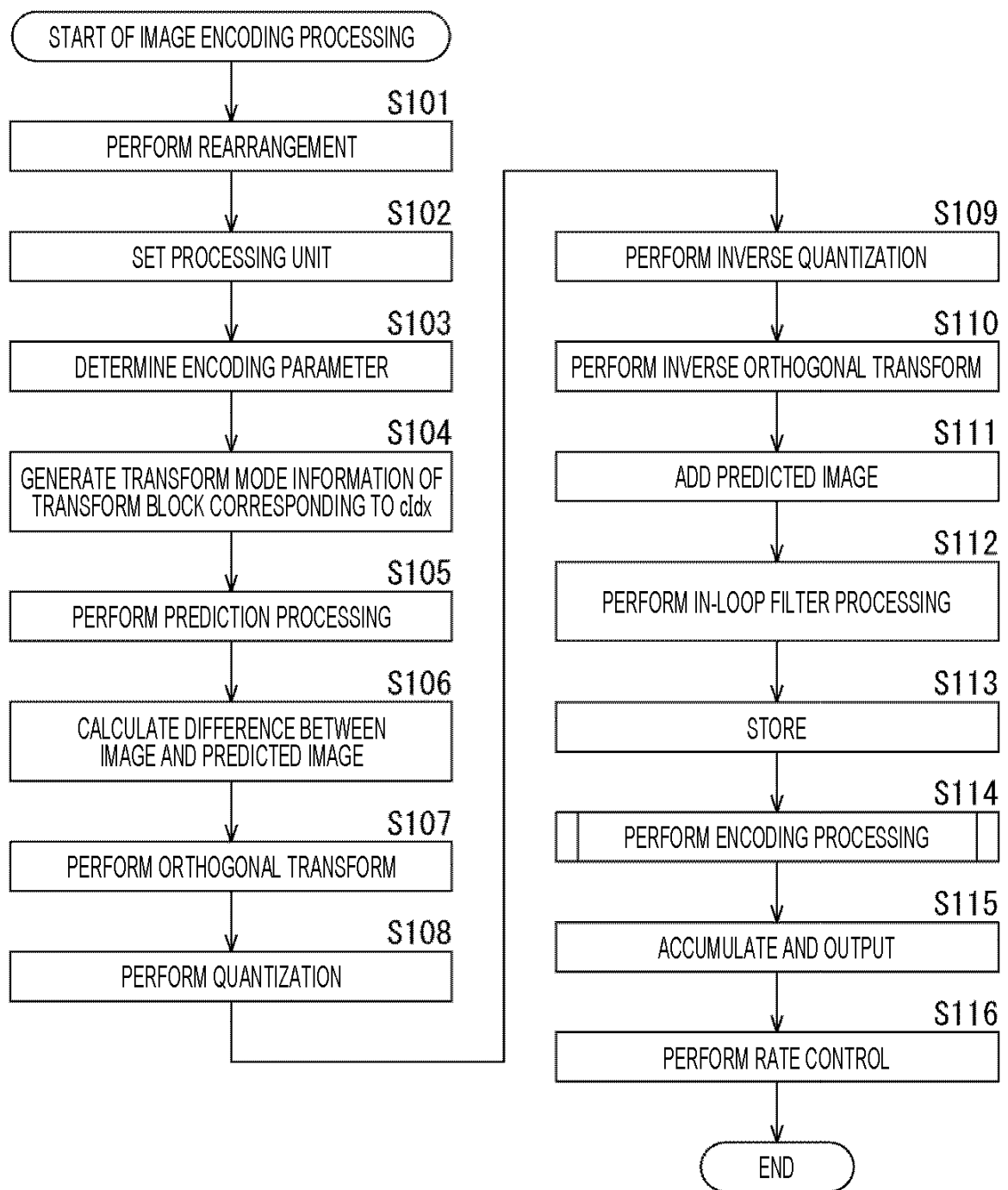


FIG. 4



5/29

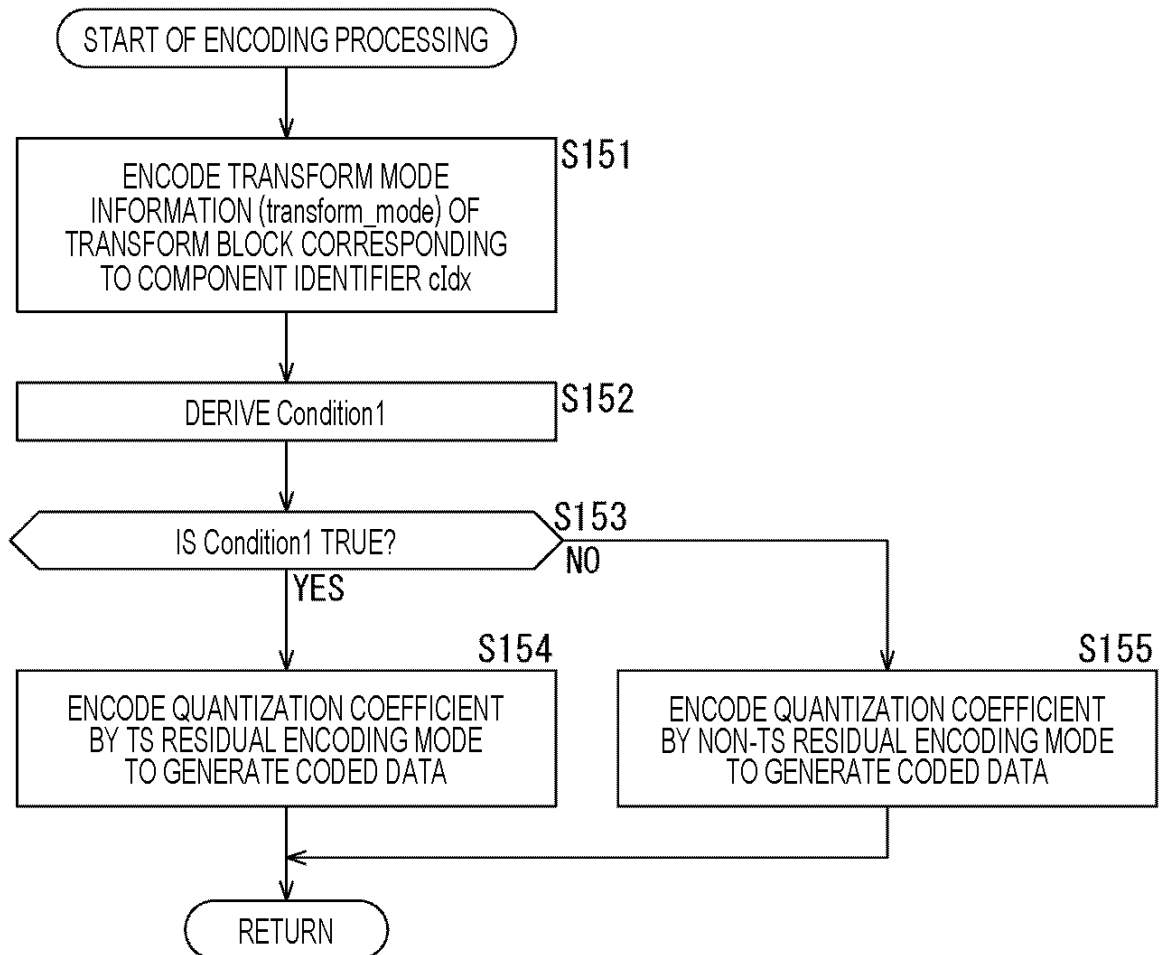
FIG. 5

FIG. 6

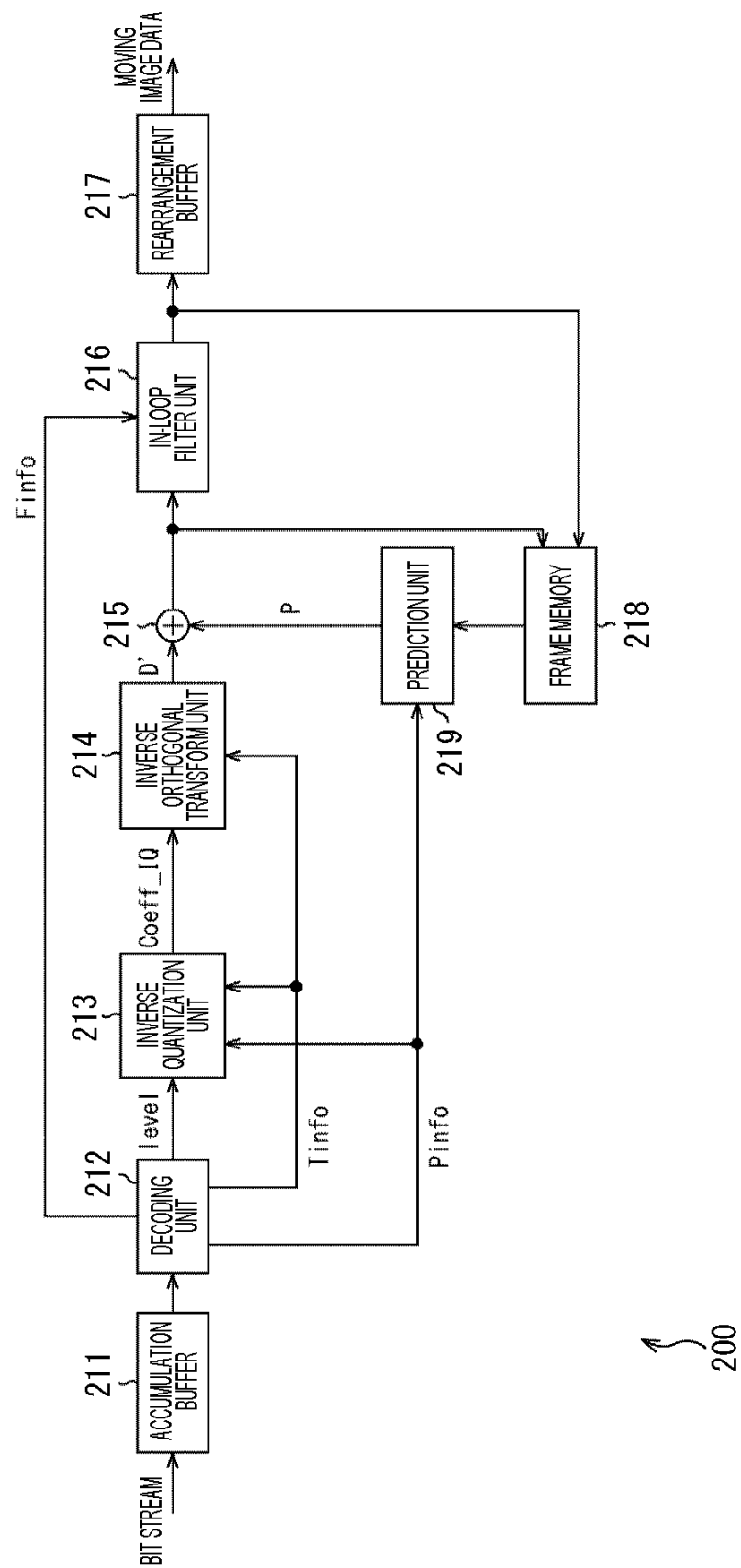
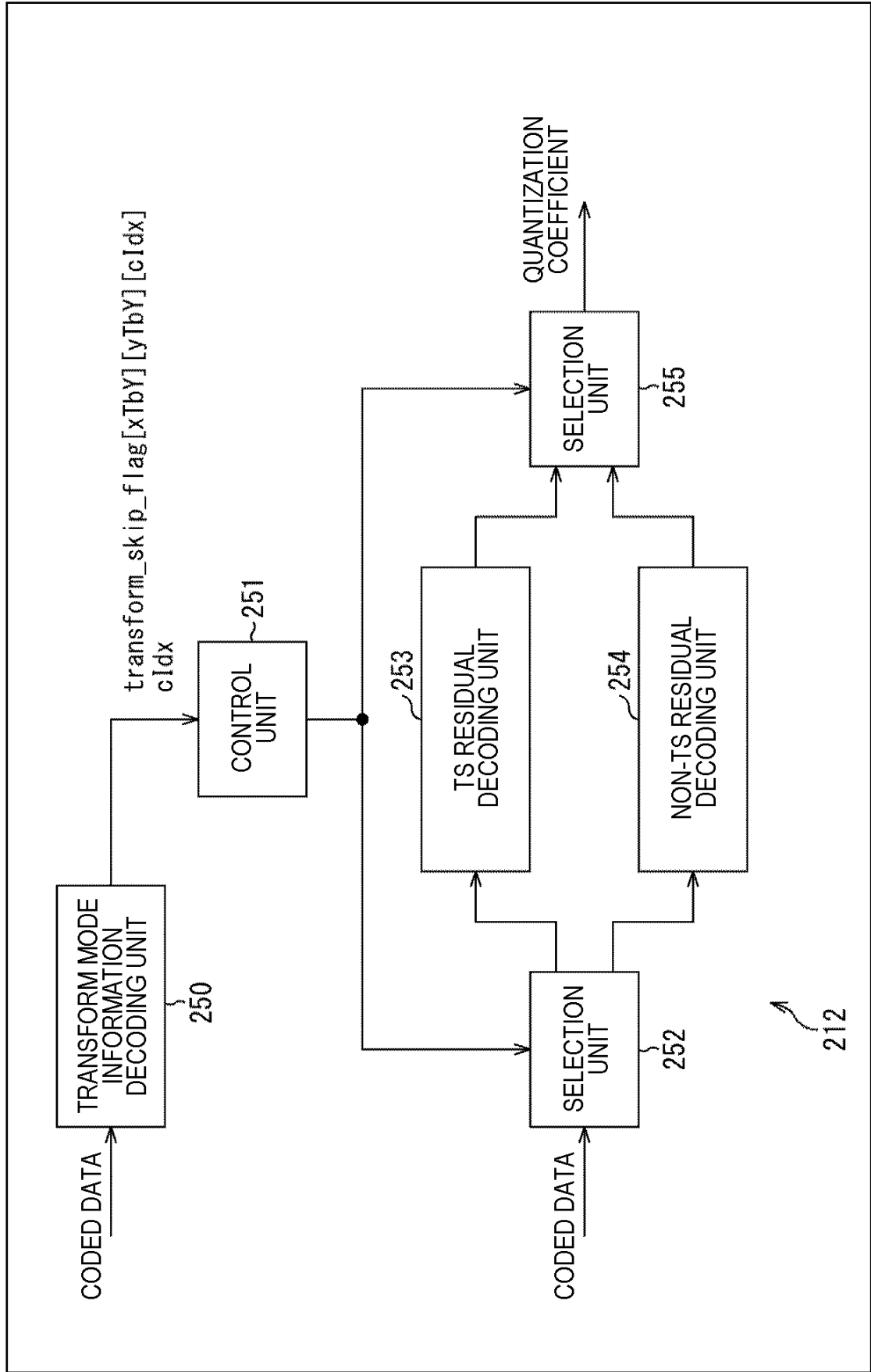
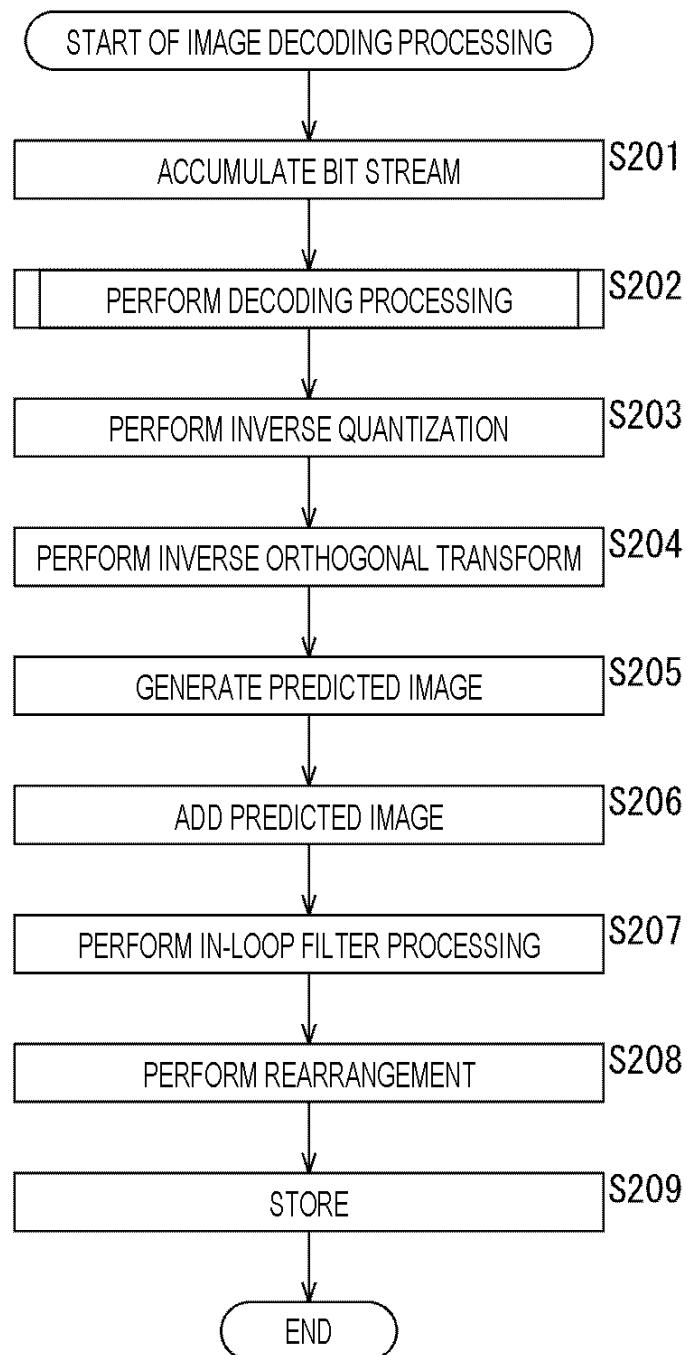


FIG. 7



8/29

FIG. 8

9/29

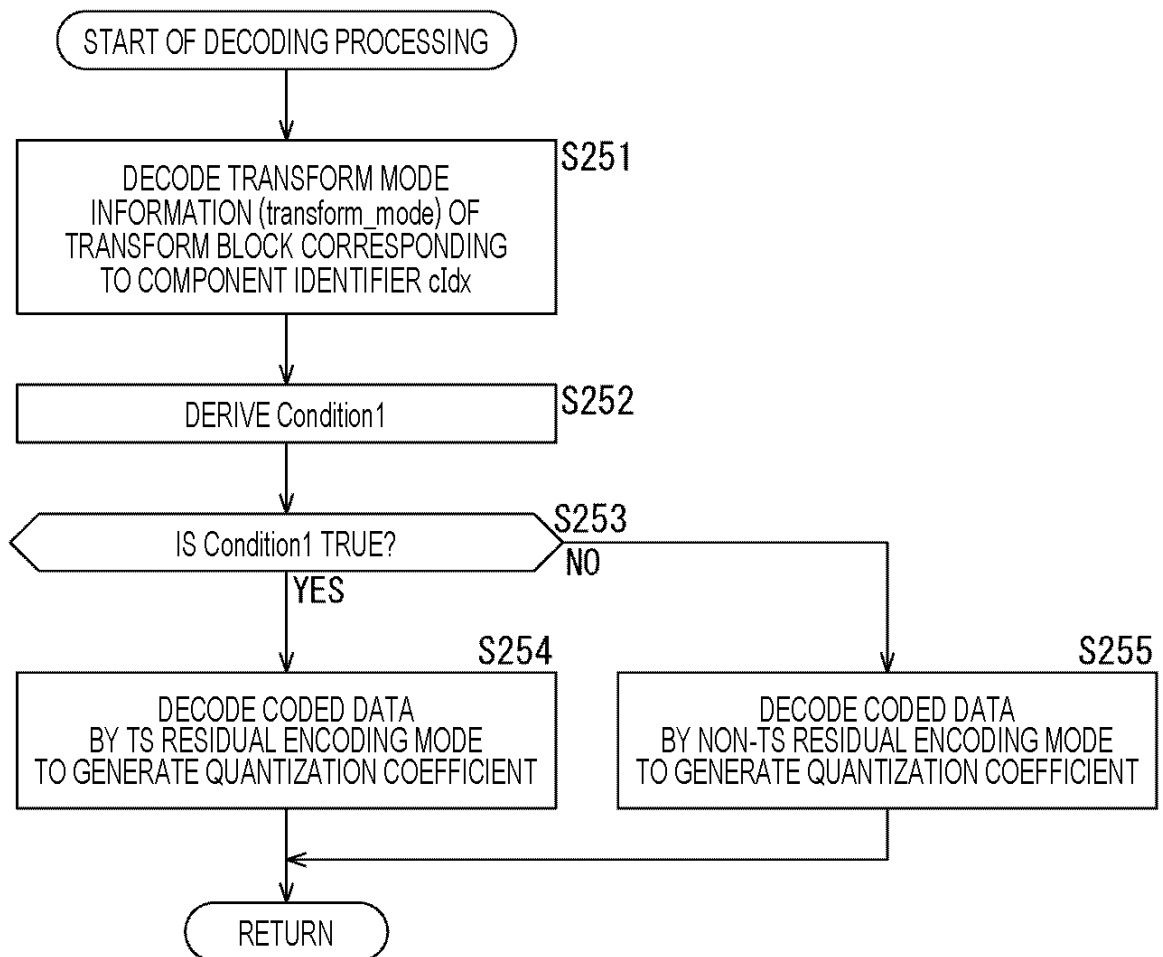
FIG. 9

FIG. 10

transform_unit(x0, y0, tbWidth, tbHeight, treeType, subTulIndex, chType) {	Descriptor
...	
if(tu_cbf_luma[x0][y0]) {	
transform_mode(x0, y0, tbWidth, tbHeight, 0)	
if(!transform_skip_flag[x0][y0][0])	
residual_coding(x0, y0, Log2(tbWidth), Log2(tbHeight), 0)	
else	
residual_ts_coding(x0, y0, Log2(tbWidth), Log2(tbHeight), 0)	
}	
if(tu_cbf_cb[x0][y0]) {	
transform_mode(x0, y0, wC, wH, 1)	
if(!transform_skip_flag[x0][y0][1])	
residual_coding(xC, yC, Log2(wC), Log2(hC), 1)	
else	
residual_ts_coding(xC, yC, Log2(wC), Log2(hC), 1)	
}	
if(tu_cbf_cr[x0][y0] &&	
!(tu_cbf_cb[x0][y0] && tu_joint_cbr_residual_flag[x0][y0])) {	
transform_mode(x0, y0, wC, hC, 2)	
if(!transform_skip_flag[x0][y0][2])	
residual_coding(xC, yC, Log2(wC), Log2(hC), 2)	
else	
residual_ts_coding(xC, yC, Log2(wC), Log2(hC), 2)	
}	
}	

FIG. 11

transform_mode(x0, y0, tbWidth, tbHeight, cldx) {	Descriptor
<pre> if (sps_transform_skip_enabled_flag && (tbWidth <= MaxTsSize) && (tbHeight <= MaxTsSize) && !(BdpcmFlag[x0][y0] && cldx==0) && (IntraSubPartitionsSplit[x0][y0] == ISP_NO_SPLIT cldx!=0) && (!cu_sbt_flag)) transform_skip_flag[x0][y0][cldx] </pre>	
<pre> if ((cldx==0) && (tbWidth <= 32) && (tbHeight <= 32) && (IntraSubPartitionsSplit[x0][y0] == ISP_NO_SPLIT) && (!cu_sbt_flag) && ((CuPredMode[chType][x0][y0] == MODE_INTER && sps_explicit_mts_inter_enabled_flag) (CuPredMode[chType][x0][y0] == MODE_INTRA && sps_explicit_mts_intra_enabled_flag) && !trasnfrom_skip_flag[x0][y0][cldx]) mts_idx [x0][y0][cldx] </pre>	ae(v)
}	ae(v)

FIG. 12

residual_ts_coding(x0, y0, log2TbWidth, log2TbHeight, cldx) {	Descriptor
log2SbSize = (Min(log2TbWidth, log2TbHeight) < 2 ? 1 : 2)	
numSbCoeff = 1 << (log2SbSize << 1)	
lastSubBlock = (1 << (log2TbWidth + log2TbHeight - 2 * log2SbSize)) - 1	
inferSbCbf = 1	
MaxCcbs = ((1 << log2TbWidth) * (1 << log2TbHeight)) * 7 >> 2	
...	
}	

FIG. 13

transform_skip_flag[x0][y0][cldx] specifies whether a transform is applied to the transform block associated with cldx or not. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture.

transform_skip_flag[x0][y0][cldx] equal to 1 specifies that no transform is applied to the associated transform block. **transform_skip_flag[x0][y0][cldx]** equal to 0 specifies that the decision whether transform is applied to the associated transform block or not depends on other syntax elements.

When **transform_skip_flag[x0][y0][cldx]** is not present, it is inferred as follows:

- If **BdcpmFlag[x0][y0]** is equal to 1 and **cldx** is equal to 0, **transform_skip_flag[x0][y0][cldx]** is inferred to be equal to 1.
- Else if **BdcpmFlag[x0][y0]** is equal to 0 and **cldx** is equal to 0, **transform_skip_flag[x0][y0][cldx]** is inferred to be equal to 0.
- Else if **BdcpmChromaFlag[x0][y0]** is equal to 1 and **cldx** is not equal to 0, **transform_skip_flag[x0][y0][cldx]** is inferred to be equal to 1.
- Else if **BdcpmChromaFlag[x0][y0]** is equal to 0 and **cldx** is not equal to 0, **transform_skip_flag[x0][y0][cldx]** is inferred to be equal to 0.

A

mts_idx[x0][y0][cldx] specifies which transform kernels are applied to the residual samples along the horizontal and vertical direction of the associated transform block with cldx. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture.

When **mts_idx[x0][y0][cldx]** is not present, it is inferred to be equal to 0.

B

FIG. 14

Descriptor	
	if(tu_cbf_luma[x0][y0] && treeType != DUAL_TREE_CHROMA){
	if (sps_transform_skip_enabled_flag && !BdpcmFlag[x0][y0][0] &&
	tbWidth <= MaxTsSize && tbHeight <= MaxTsSize && (
	IntraSubPartitionsSplit[x0][y0] == ISP_NO_SPLIT) && !cu_sbt_flag)
ae(v)	transform_skip_flag[x0][y0][0]
	if(!transform_skip_flag[x0][y0][0])
	residual_coding(x0, y0, Log2(tbWidth), Log2(tbHeight), 0)
ae(v)	else
	residual_ts_coding(x0, y0, Log2(tbWidth), Log2(tbHeight), 0)
	}
	if(tu_cbf_cb[xC][yC] && treeType != DUAL_TREE_LUMA){
	if (sps_transform_skip_enabled_flag && !BdpcmFlag[x0][y0][1] &&
	wC <= MaxTsSize && hC <= MaxTsSize && !cu_sbt_flag)
	transform_skip_flag[xC][yC][1]
	if(!transform_skip_flag[xC][yC][1])
	residual_coding(xC, yC, Log2(wC), Log2(hC), 1)
	else
	residual_ts_coding(xC, yC, Log2(wC), Log2(hC), 1)
	if (sps_transform_skip_enabled_flag && !BdpcmFlag[x0][y0][2] &&
	wC <= MaxTsSize && hC <= MaxTsSize && !cu_sbt_flag)
	transform_skip_flag[xC][yC][2]
	if(!transform_skip_flag[xC][yC][2])
	residual_coding(xC, yC, Log2(wC), Log2(hC), 2)
	else
	residual_ts_coding(xC, yC, Log2(wC), Log2(hC), 2)
	}

FIG. 15

sequence_parameter_set_rbsp(){	
...	
sps_transform_skip_enabled_flag	u(1)
if(sps_transform_skip_enabled_flag)	
sps_bdpcm_enabled_flag	u(1)
sps_ts_residual_coding_use_flag	u(1)
...	

FIG. 16

transform_unit(x0, y0, tbWidth, tbHeight, treeType, subTulIndex, chType) {	Descriptor
...	
if(tu_cbf_luma[x0][y0]) {	
transform_mode(x0, y0, tbWidth, tbHeight, 0)	
if(!(transform_skip_flag[x0][y0][0] == IS_TS && sps_ts_residual_coding_use_flag)	
)	
residual_coding(x0, y0, Log2(tbWidth), Log2(tbHeight), 0)	
else	
residual_ts_coding(x0, y0, Log2(tbWidth), Log2(tbHeight), 0)	
}	
if(tu_cbf_cb[x0][y0])	
transform_mode(x0, y0, wC, wH, 1)	
if(!(transform_skip_flag[x0][y0][1] == IS_TS && sps_ts_residual_coding_use_flag)	
residual_coding(xC, yC, Log2(wC), Log2(hC), 1)	
else	
residual_ts_coding(xC, yC, Log2(wC), Log2(hC), 1)	
if(tu_cbf_cr[x0][y0] &&	
!(tu_cbf_cb[x0][y0] && tu_joint_cbr_residual_flag[x0][y0])) {	
transform_mode(x0, y0, wC, hC, 2)	
if(!(transform_skip_flag[x0][y0][2] == IS_TS && sps_ts_residual_coding_use_flag))	
residual_coding(xC, yC, Log2(wC), Log2(hC), 2)	
else	
residual_ts_coding(xC, yC, Log2(wC), Log2(hC), 2)	
}	
}	

FIG. 17

sps_ts_residual_coding_use_flag equal to 1 specifies that **ts_residual_coding** is used for residual coding when **transform_skip_flag** corresponding to **component_id** is equal to "IS_SKIP".
sps_ts_residual_coding_use_flag equal to 0 specifies that **ts_residual_coding** is not used for residual coding when **transform_skip_flag** corresponding to **component_id** is equal to "IS_SKIP".

"IS_SKIP" means "transform skip is used"

FIG. 18

sequence_parameter_set_rbsp(){	
...	
sps_transform_skip_enabled_flag	u(1)
if(sps_transform_skip_enabled_flag)	
sps_bdpcm_enabled_flag	u(1)
sps_ts_residual_coding_use_for_bdpcm_flag	u(1)
...	

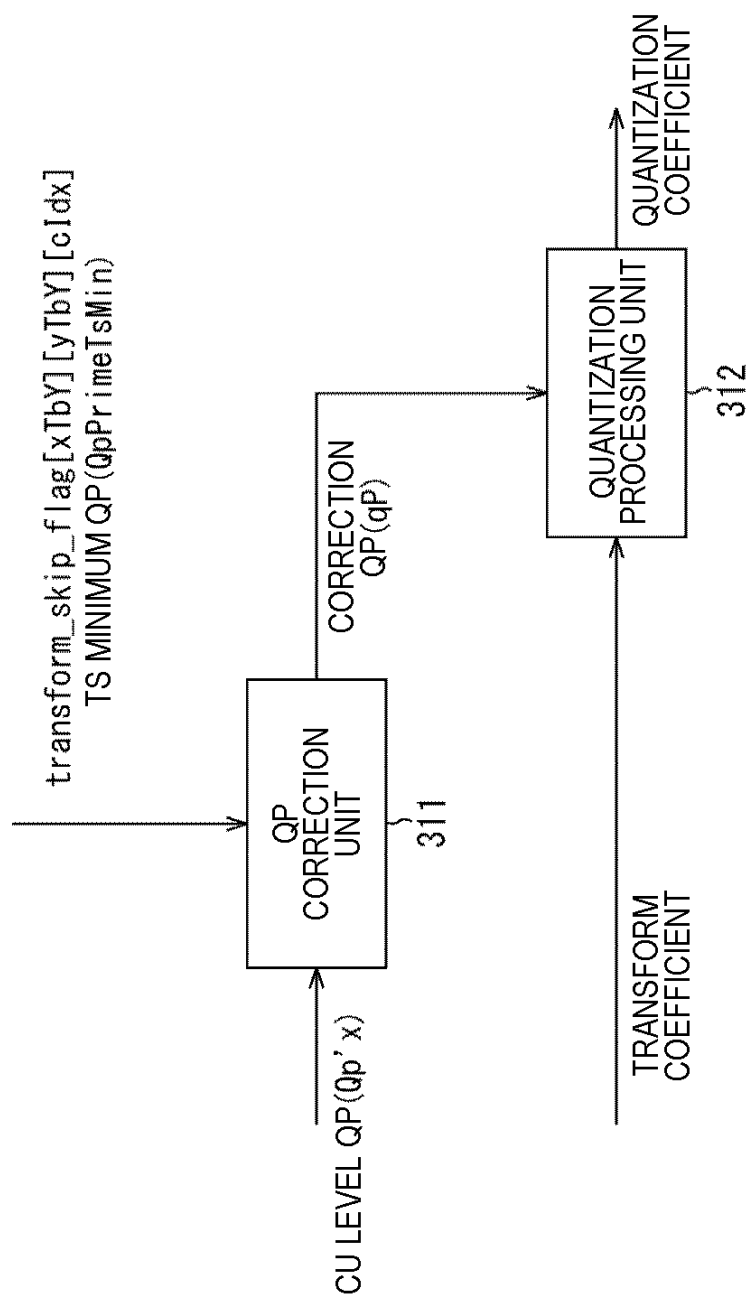
FIG. 19

transform_unit(x0, y0, tbWidth, tbHeight, treeType, subTuIndex, chType) {	Descriptor
...	
if(tu_cbf_luma[x0][y0]) {	
transform_mode(x0, y0, tbWidth, tbHeight, 0)	
if(!(transform_skip_flag[x0][y0][0] == IS_TS) (intra_bdpcm_flag &&	
sps_ts_residual_coding_use_for_bdpcm_flag==0))	
residual_coding(x0, y0, Log2(tbWidth), Log2(tbHeight), 0)	
else	
residual_ts_coding(x0, y0, Log2(tbWidth), Log2(tbHeight), 0)	
}	
if(tu_cbf_cb[x0][y0])	
transform_mode(x0, y0, wC, wH, 1)	
if(!(transform_skip_flag[x0][y0][1] == IS_TS) (intra_bdpcm_chroma_flag &&	
sps_ts_residual_coding_use_for_bdpcm_flag==0))	
residual_coding(xC, yC, Log2(wC), Log2(hC), 1)	
else	
residual_ts_coding(xC, yC, Log2(wC), Log2(hC), 1)	
if(tu_cbf_cr[x0][y0] &&	
!(tu_cbf_cb[x0][y0] && tu_joint_cbr_residual_flag[x0][y0])) {	
transform_mode(x0, y0, wC, hC, 2)	
if(!(transform_skip_flag[x0][y0][2] == IS_TS) (intra_bdpcm_chroma_flag &&	
sps_ts_residual_coding_use_for_bdpcm_flag==0))	
residual_coding(xC, yC, Log2(wC), Log2(hC), 2)	
else	
residual_ts_coding(xC, yC, Log2(wC), Log2(hC), 2)	
}	
}	

FIG. 20

sps_ts_residual_coding_use_for_bdpcm_flag equal to 1 specifies that **ts_residual_coding** is used for residual coding when **bdpcm** mode is used. **sps_ts_residual_coding_use_for_bdpcm_flag** equal to 0 specifies that **ts_residual_coding** is not used for residual coding when **bdpcm** mode is used.

FIG. 21



114

22/29

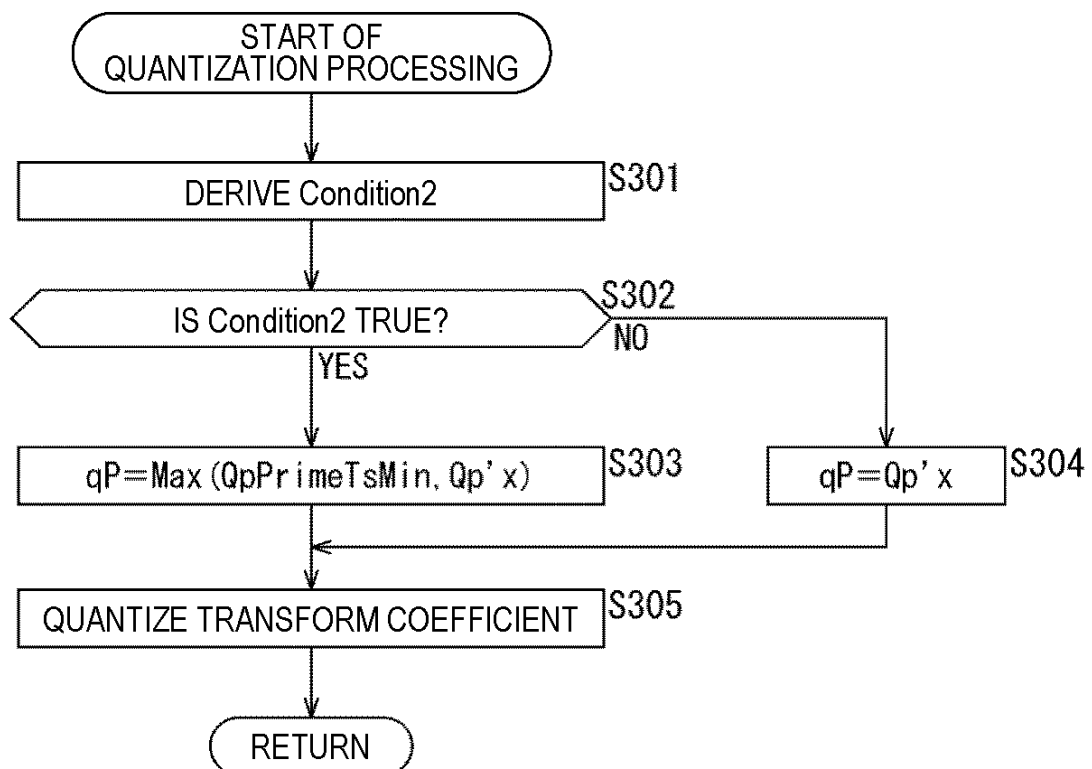
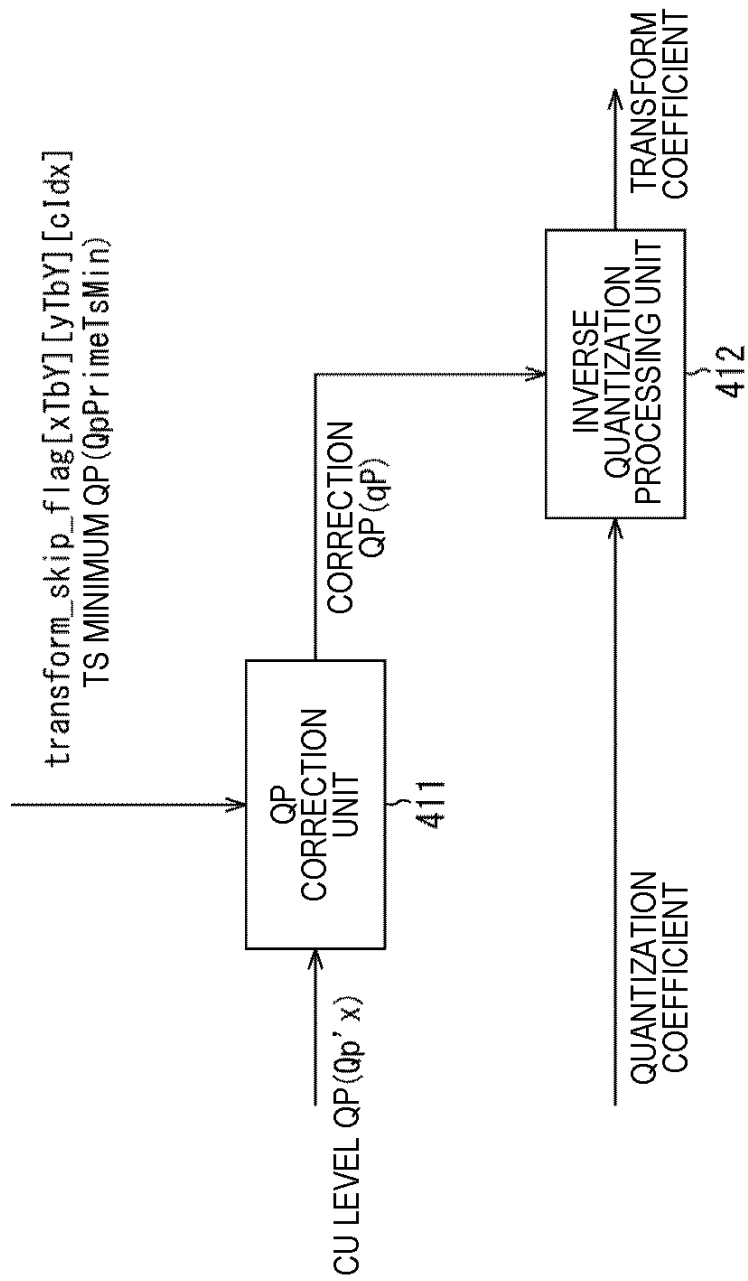
FIG. 22

FIG. 23



213

24/29

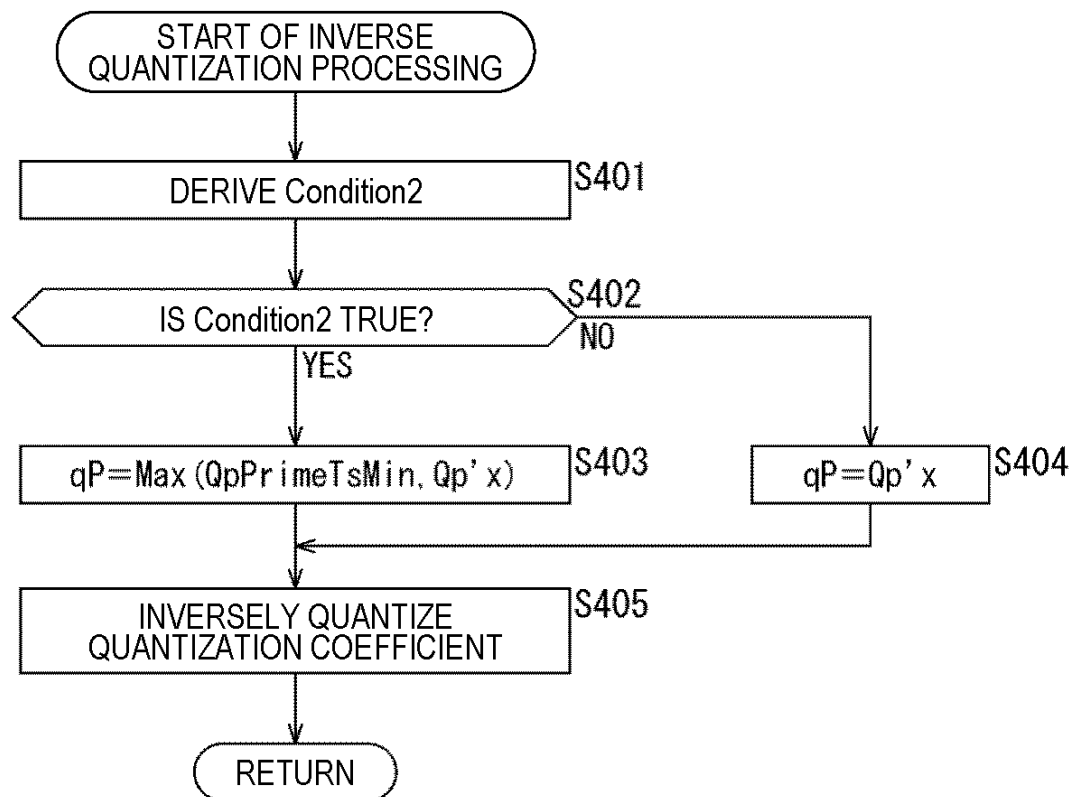
FIG. 24

FIG. 25

The quantization parameter qP is derived as follows:

- If $cIdx$ is equal to 0 and $transform_skip_flag[xTbY][yTbY][0]$ is equal to 0, the following applies:
 $qP = Qp'_{-Y}$
- Otherwise, if $cIdx$ is equal to 0 (and $transform_skip_flag[xTbY][yTbY][0]$ is equal to 1), the following applies:
 $qP = \text{Max}(QpPrimeTsMin, Qp'_{-Y})$
- Otherwise, if $TuCRsModel[xTbY][yTbY]$ is equal to 2, the following applies:
 $qP = (transform_skip_flag[xTbY][yTbY][1]) \neq IS_SKIP ? Qp'_{-CbCr} : \text{Max}(QpPrimeTsMin, Qp'_{-CbCr})$
- Otherwise, if $cIdx$ is equal to 1, the following applies:
 $qP = (transform_skip_flag[xTbY][yTbY][1]) \neq IS_SKIP ? Qp'_{-Cb} : \text{Max}(QpPrimeTsMin, Qp'_{-Cb})$
- Otherwise ($cIdx$ is equal to 2), the following applies:
 $qP = (transform_skip_flag[xTbY][yTbY][2]) \neq IS_SKIP ? Qp'_{-Cr} : \text{Max}(QpPrimeTsMin, Qp'_{-Cr})$

The variable $rectNonTsFlag$ is derived as follows:

$$rectNonTsFlag = ((\text{Log2}(nTbW) + \text{Log2}(nTbH)) \& 1) = 1 \& \text{transform_skip_flag}[xTbY][yTbY][cIdx] = 0)$$

The variables $bdShift$, $rectNorm$ and $bdOffset$ are derived as follows:

$$bdShift = \text{bitDepth} + ((rectNonTsFlag ? 1 : 0) + (\text{Log2}(nTbW) + \text{Log2}(nTbH)) / 2) ? 5 + dep_quant_enabled_flag$$

$$bdOffset = (1 << bdShift) >> 1$$

FIG. 26

The variable ctxInc is derived as follows:

- If transform_skip_flag[xS][yS][cldx] is equal to 1, the following applies:
ctxInc = 90 + locSumSig
- Otherwise (transform_skip_flag[xS][yS][cldx] is equal to 1), the following applies:
 - If cldx is equal to 0, ctxInc is derived as follows:
ctxInc = 12 * Max(0, QState ? 1) + Min((locSumAbsPass1 + 1) >> 1, 3) +
(d < 2 ? 8 : (d < 5 ? 4 : 0))
 - Otherwise (cldx is greater than 0), ctxInc is derived as follows:
ctxInc = 36 + 8 * Max(0, QState ? 1) + Min((locSumAbsPass1 + 1) >> 1, 3) + (d < 2 ? 4 : 0)

FIG. 27

- The variable csbCtx is initialized with 0 and modified as follows:
 - If transform_skip_flag[xS][yS][cldx] is equal to 1, the following applies:
 - When xS is greater than 0, csbCtx is modified as follows:

$$\text{csbCtx} += \text{coded_sub_block_flag}[xS ? 1][yS]$$
 - When yS is greater than 0, csbCtx is modified as follows:

$$\text{csbCtx} += \text{coded_sub_block_flag}[xS][yS ? 1]$$
 - Otherwise (transform_skip_flag[xS][yS][cldx] is equal to 0), the following applies:
 - When xS is less than $(1 \ll (\log_2 \text{TbWidth} ? \log_2 \text{SbWidth})) ? 1$, csbCtx is modified as follows:

$$\text{csbCtx} += \text{coded_sub_block_flag}[xS + 1][yS]$$
 - When yS is less than $(1 \ll (\log_2 \text{TbHeight} ? \log_2 \text{SbHeight})) ? 1$, csbCtx is modified as follows:

$$\text{csbCtx} += \text{coded_sub_block_flag}[xS][yS + 1]$$

The context index increment ctxInc is derived using the colour component index cldx and csbCtx as follows:

- If transform_skip_flag[xS][yS][cldx] is equal to 1, the following applies:

$$\text{ctxInc} = 4 + \text{csbCtx}$$
- Otherwise (transform_skip_flag[xS][yS][cldx] is equal to 0), the following applies:
 - If cldx is equal to 0, ctxInc is derived as follows:

$$\text{ctxInc} = \text{Min}(\text{csbCtx}, 1)$$
- Otherwise (cldx is greater than 0), ctxInc is derived as follows:

$$\text{ctxInc} = 2 + \text{Min}(\text{csbCtx}, 1)$$

FIG. 28

Syntax element	binidx				
	0	1	2	3	4
coeff_sign_flag[] transform_skip_flag[x0][y0][cldx] = 0	bypass	na	na	na	na
coeff_sign_flag[] transform_skip_flag[x0][y0][cldx] = 1	(MaxCcbs > 0) ? (0..5) : bypass	na	na	na	na

FIG. 29

