FIG.1

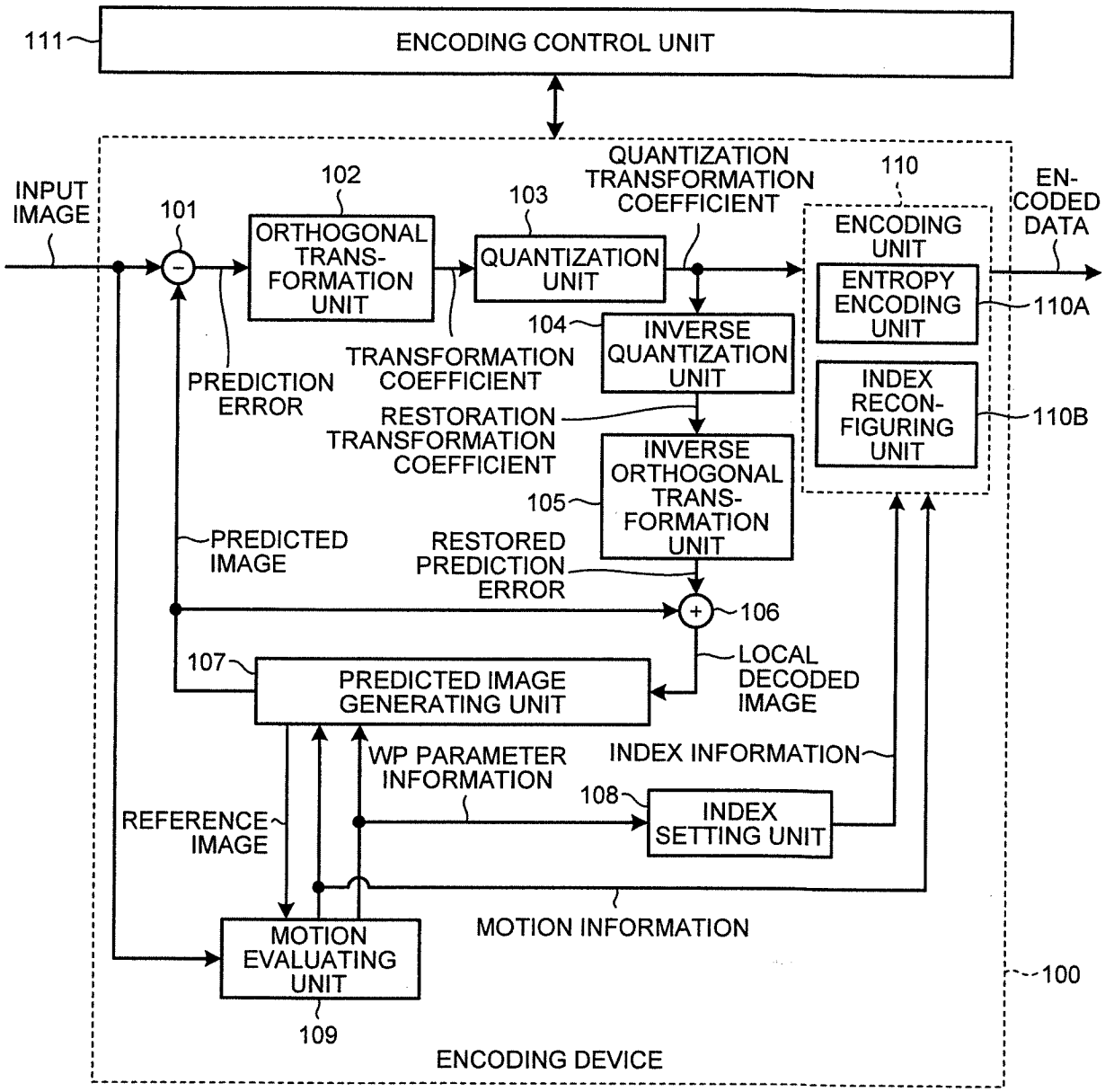

111
ENCODING CONTROL UNIT
INPUT IMAGE
101
102
ORTHOGONAL TRANS-FORMATION UNIT
103
QUANTIZATION UNIT
QUANTIZATION TRANSFORMATION COEFFICIENT
110
ENCODING UNIT
ENTROPY ENCODING UNIT
110A
EN-CODED DATA
INDEX RECON-FIGURING UNIT
110B
104
INVERSE QUANTIZATION UNIT
PREDICTION ERROR
TRANSFORMATION COEFFICIENT
RESTORATION TRANSFORMATION COEFFICIENT
105
INVERSE ORTHOGONAL TRANS-FORMATION UNIT
PREDICTED IMAGE
RESTORED PREDICTION ERROR
106
LOCAL DECODED IMAGE
107
PREDICTED IMAGE GENERATING UNIT
WP PARAMETER INFORMATION
INDEX INFORMATION
108
INDEX SETTING UNIT
REFERENCE IMAGE
MOTION INFORMATION
MOTION EVALUATING UNIT
109
100
ENCODING DEVICE

FIG.2

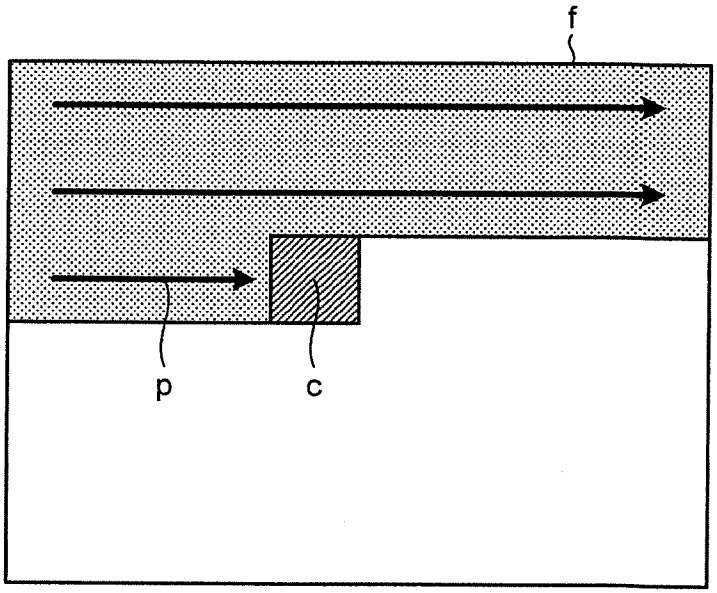


FIG.3A

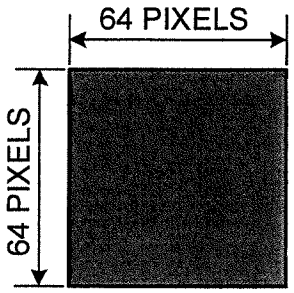


FIG.3B

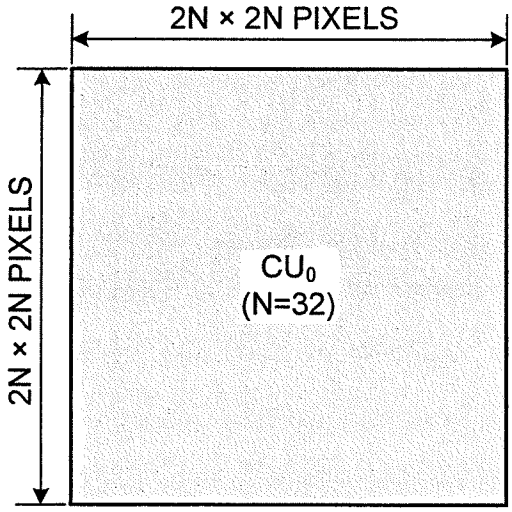


FIG.3C

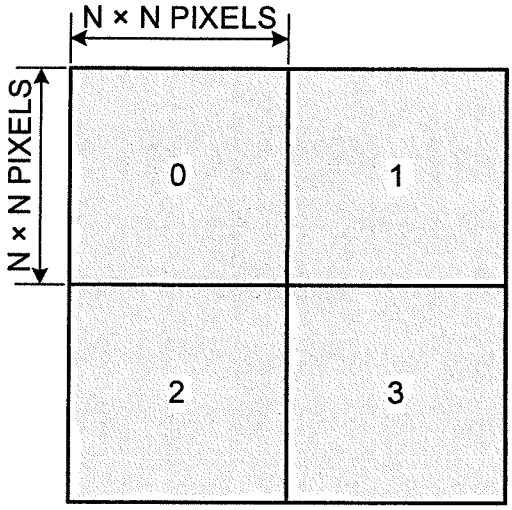


FIG.3D

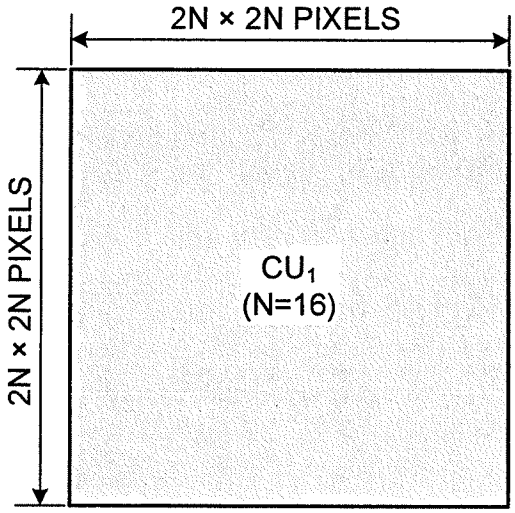


FIG.4

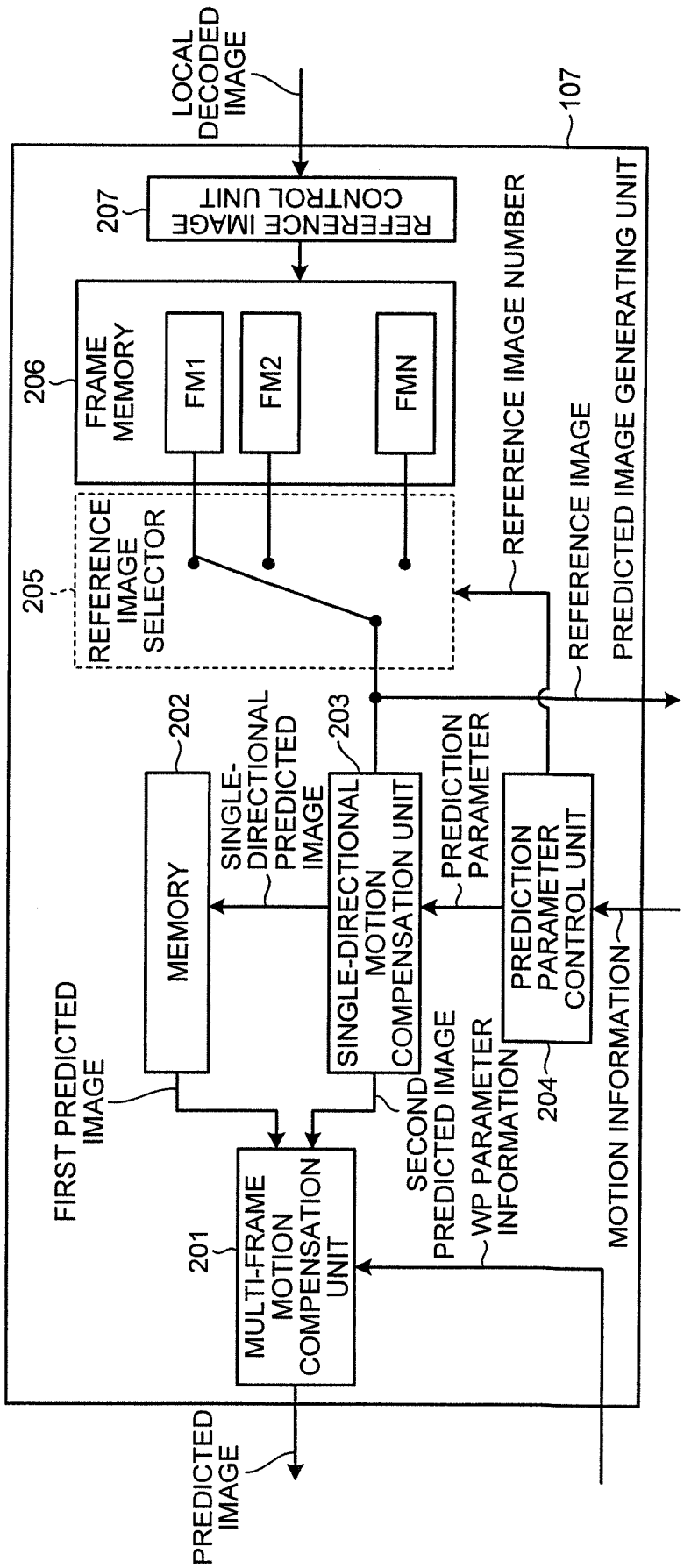

LOCAL DECODED IMAGE
107
207
REFERENCE IMAGE CONTROL UNIT
206
FRAME MEMORY
FM1
FM2
FMN
205
REFERENCE IMAGE SELECTOR
REFERENCE IMAGE NUMBER
REFERENCE IMAGE
PREDICTED IMAGE GENERATING UNIT
202
MEMORY
SINGLE-DIRECTIONAL PREDICTED IMAGE
203
SINGLE-DIRECTIONAL MOTION COMPENSATION UNIT
PREDICTION PARAMETER
PREDICTION PARAMETER CONTROL UNIT
FIRST PREDICTED IMAGE
SECOND PREDICTED IMAGE
WP PARAMETER INFORMATION
204
MOTION INFORMATION
201
MULTI-FRAME MOTION COMPENSATION UNIT
PREDICTED IMAGE

FIG.5

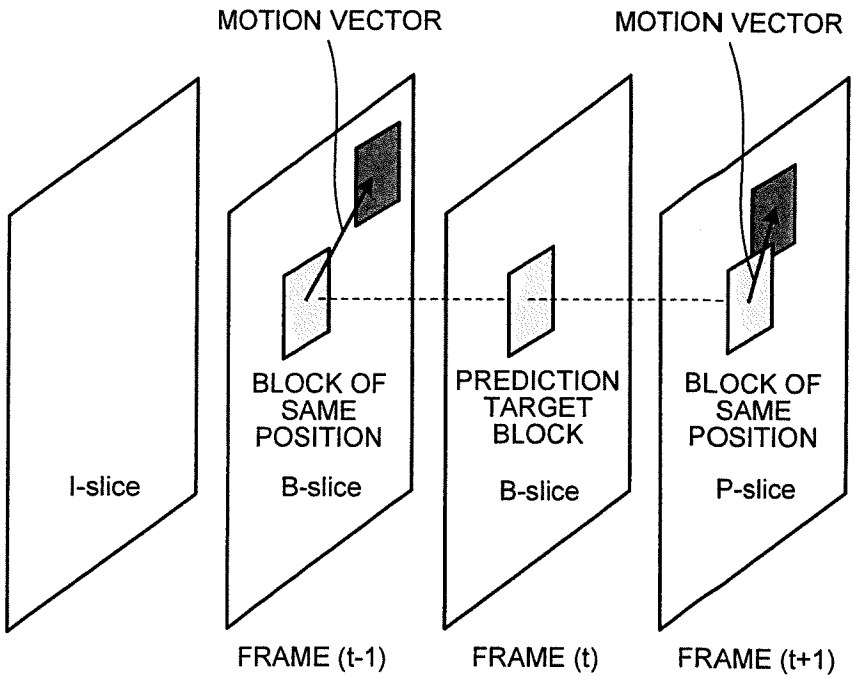

MOTION VECTOR
MOTION VECTOR
I-slice
BLOCK OF SAME POSITION
B-slice
PREDICTION TARGET BLOCK
B-slice
BLOCK OF SAME POSITION
P-slice
FRAME (t-1)
FRAME (t)
FRAME (t+1)

FIG.6

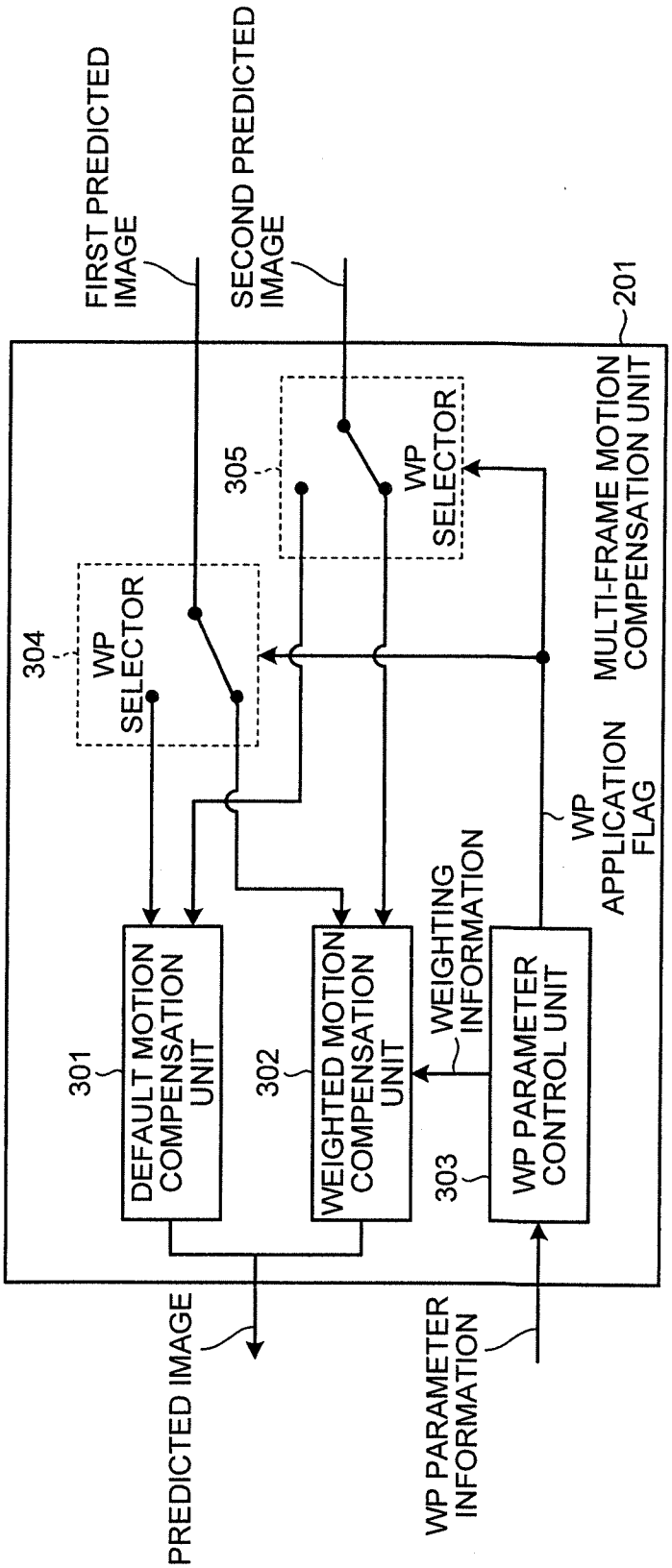

FIRST PREDICTED IMAGE
SECOND PREDICTED IMAGE
201
MULTI-FRAME MOTION COMPENSATION UNIT
304
WP SELECTOR
305
WP SELECTOR
WP APPLICATION FLAG
301
DEFAULT MOTION COMPENSATION UNIT
302
WEIGHTED MOTION COMPENSATION UNIT
WEIGHTING INFORMATION
303
WP PARAMETER CONTROL UNIT
PREDICTED IMAGE
WP PARAMETER INFORMATION

FIG.7

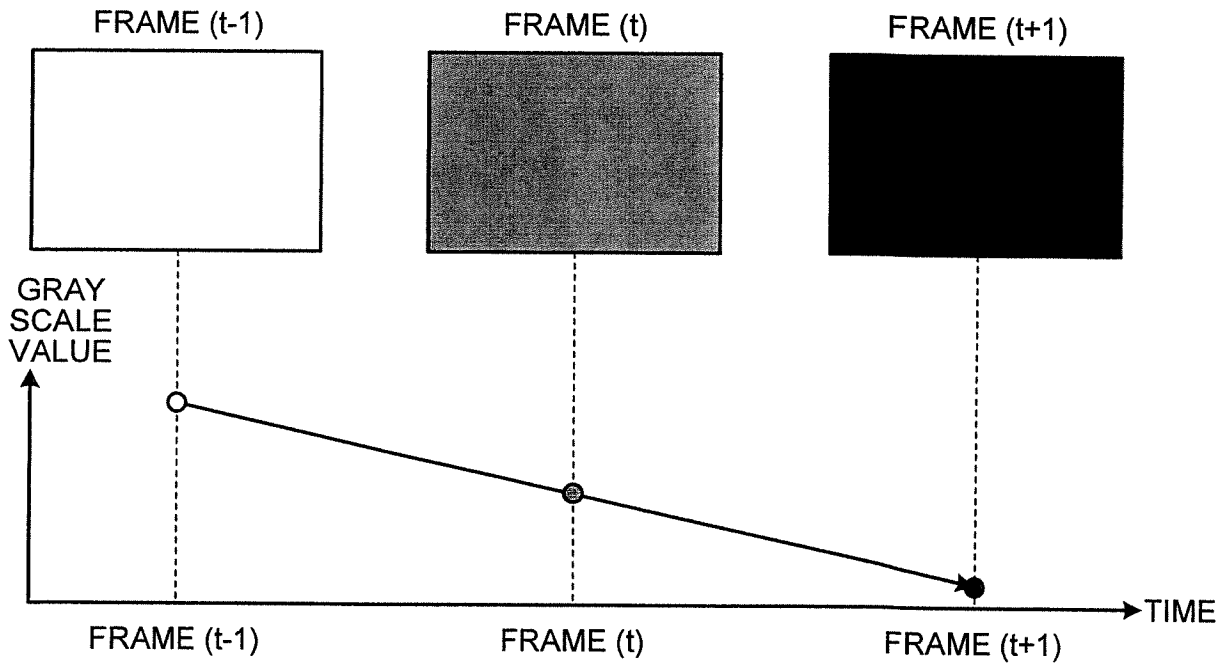

FRAME (t-1)
FRAME (t)
FRAME (t+1)
GRAY SCALE VALUE
TIME
FRAME (t-1)
FRAME (t)
FRAME (t+1)

FIG.8A

LIST NUMBER	0						
REFERENCE NUMBER	0	1	2	3	4	...	N
WP APPLICATION FLAG	WP_flag[0]	WP_flag[1]	WP_flag[2]	WP_flag[3]	WP_flag[4]	...	WP_flag[N]
WEIGHTING FACTOR	Weight[0]	Weight[1]	Weight[2]	Weight[3]	Weight[4]	...	Weight[N]
OFFSET	Offset[0]	Offset[1]	Offset[2]	Offset[3]	Offset[4]	...	Offset[N]

FIG.8B

LIST NUMBER	1						
REFERENCE NUMBER	0	1	2	3	4	...	N
WP APPLICATION FLAG	WP_flag[0]	WP_flag[1]	WP_flag[2]	WP_flag[3]	WP_flag[4]	...	WP_flag[N]
WEIGHTING FACTOR	Weight[0]	Weight[1]	Weight[2]	Weight[3]	Weight[4]	...	Weight[N]
OFFSET	Offset[0]	Offset[1]	Offset[2]	Offset[3]	Offset[4]	...	Offset[N]

FIG.9

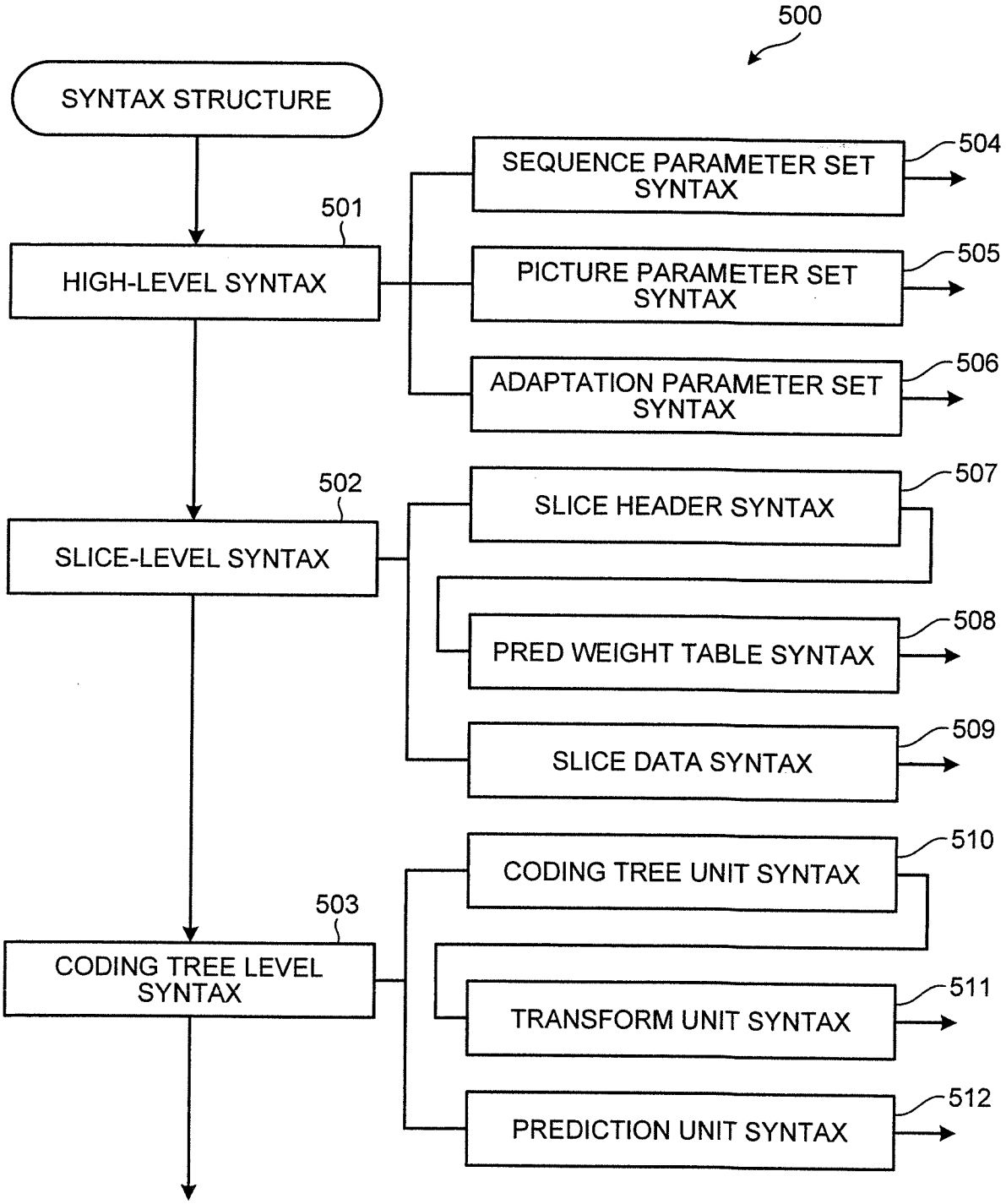

500
SYNTAX STRUCTURE
501
HIGH-LEVEL SYNTAX
502
SLICE-LEVEL SYNTAX
503
CODING TREE LEVEL SYNTAX
SEQUENCE PARAMETER SET SYNTAX
504
PICTURE PARAMETER SET SYNTAX
505
ADAPTATION PARAMETER SET SYNTAX
506
SLICE HEADER SYNTAX
507
PRED WEIGHT TABLE SYNTAX
508
SLICE DATA SYNTAX
509
CODING TREE UNIT SYNTAX
510
TRANSFORM UNIT SYNTAX
511
PREDICTION UNIT SYNTAX
512

FIG.10

pic_parameter_set_rbsp( ) {	**Descriptor**
pic_parameter_set_id	u (8)
seq_parameter_set_id	u (8)
...	
num_ref_idx_l0_default_active_minus1	ue (v)
num_ref_idx_l1_default_active_minus1	ue (v)
weighted_pred_flag	u (1)
weighted_bipred_idc	u (2)
...	
rbsp_trailing_bits( )	
}	

FIG.11

slice_header( ) {	**Descriptor**
...	
slice_type	ue (v)
pic_parameter_set_id	ue (v)
...	
if(slice_type = = P || slice_type = = B) {	
num_ref_idx_active_override_flag	u (1)
if(num_ref_idx_active_override_flag) {	
num_ref_idx_l0_active_minus1	ue (v)
if(slice_type = = B)	
num_ref_idx_l1_active_minus1	ue (v)
}	
}	
...	
if((weighted_pred_flag && (slice_type = = P)) || (weighted_bipred_idc = = 1 && slice_type = = B))	
pred_weight_table( )	
...	
}	

FIG.12

pred_weight_table( ) {	**Descriptor**
luma_log2_weight_denom	ue (v)
if(chroma_format_idc != MONO_IDX)	
chroma_log2_weight_denom	ue (v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {	
luma_weight_l0_flag	u (1)
if(luma_weight_l0_flag) {	
luma_weight_l0[i]	se (v)
luma_offset_l0[i]	se (v)
}	
if (chroma_format_idc != MONO_IDX) {	
chroma_weight_l0_flag	u (1)
if(chroma_weight_l0_flag)	
for(j =0; j < COLOR_COMP - 1; j++) {	
chroma_weight_l0[i][j]	se (v)
chroma_ offset_l0[i][j]	se (v)
}	
}	
}	
if(slice_type = = B_SLICE)	
for(i = 0; i <= num_ref_idx_l1_active_minus1; i++) {	
luma_weight_l1_flag	u (1)
if(luma_weight_l1_flag) {	
luma_weight_l1[i]	se (v)
luma_ offset_l1[i]	se (v)
}	
if(chroma_format_idc != MONO_IDX) {	
chroma_weight_l1_flag	u (1)
if(chroma_weight_l1_flag) {	
for(j =0; j < COLOR_COMP - 1; j++) {	
chroma_weight_l1[i][j]	se (v)
chroma_ offset_l1[i][j]	se (v)
}	
}	
}	
}	
}	
}	

FIG.13

pred_weight_table( ) {	Descriptor
luma_log2_weight_denom	**ue (v)**
if(chroma_format_idc != MONO_IDX)	
delta_chroma_log2_weight_denom	**ue (v)**
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {	
luma_weight_l0_flag	**u (1)**
if(luma_weight_l0_flag) {	
delta_luma_weight_l0[i]	**se (v)**
delta_luma_offset_l0[i]	**se (v)**
}	
if (chroma_format_idc != MONO_IDX) {	
chroma_weight_l0_flag	**u (1)**
if(chroma_weight_l0_flag)	
for(j =0; j < COLOR_COMP - 1; j++) {	
delta_chroma_weight_l0[i][j]	**se (v)**
delta_chroma_ offset_l0[i][j]	**se (v)**
}	
}	
}	
if(slice_type = = B_SLICE)	
for(i = 0; i <= num_ref_idx_l1_active_minus1; i++) {	
luma_weight_l1_flag	**u (1)**
if(luma_weight_l1_flag) {	
delta_luma_weight_l1[i]	**se (v)**
delta_luma_ offset_l1[i]	**se (v)**
}	
if(chroma_format_idc != MONO_IDX) {	
chroma_weight_l1_flag	**u (1)**
if(chroma_weight_l1_flag) {	
for(j =0; j < COLOR_COMP - 1; j++) {	
delta_chroma_weight_l1[i][j]	**se (v)**
delta_chroma_ offset_l1[i][j]	**se (v)**
}	
}	
}	
}	
}	
}	

FIG.14

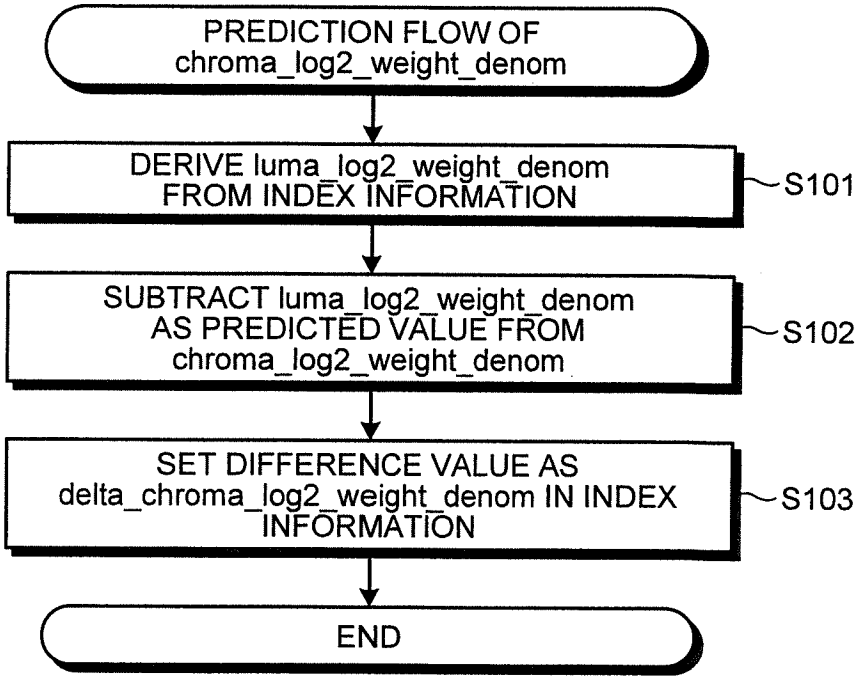

PREDICTION FLOW OF chroma_log2_weight_denom
DERIVE luma_log2_weight_denom FROM INDEX INFORMATION
S101
SUBTRACT luma_log2_weight_denom AS PREDICTED VALUE FROM chroma_log2_weight_denom
S102
SET DIFFERENCE VALUE AS delta_chroma_log2_weight_denom IN INDEX INFORMATION
S103
END

FIG.15

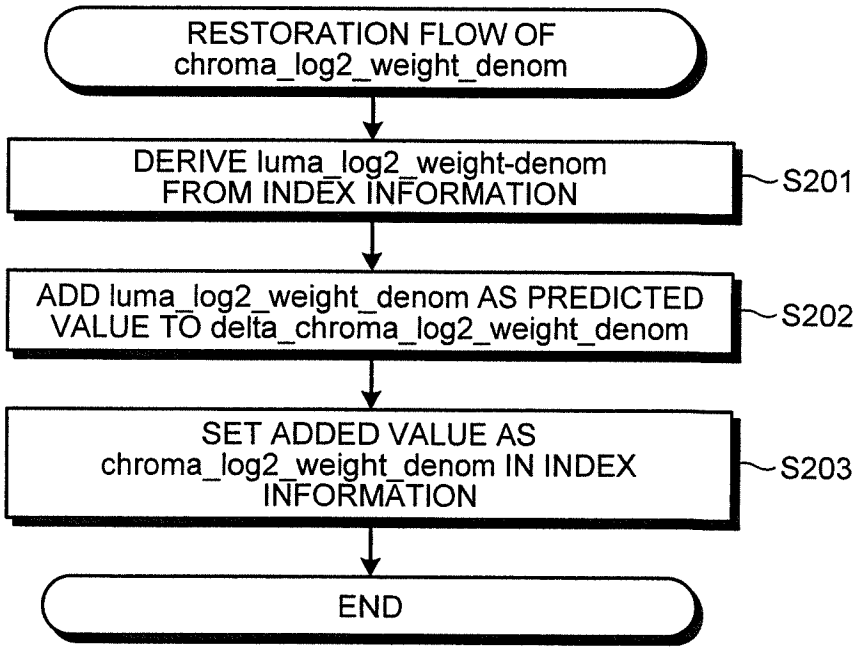

RESTORATION FLOW OF chroma_log2_weight_denom
DERIVE luma_log2_weight-denom FROM INDEX INFORMATION
S201
ADD luma_log2_weight_denom AS PREDICTED VALUE TO delta_chroma_log2_weight_denom
S202
SET ADDED VALUE AS chroma_log2_weight_denom IN INDEX INFORMATION
S203
END

FIG.16

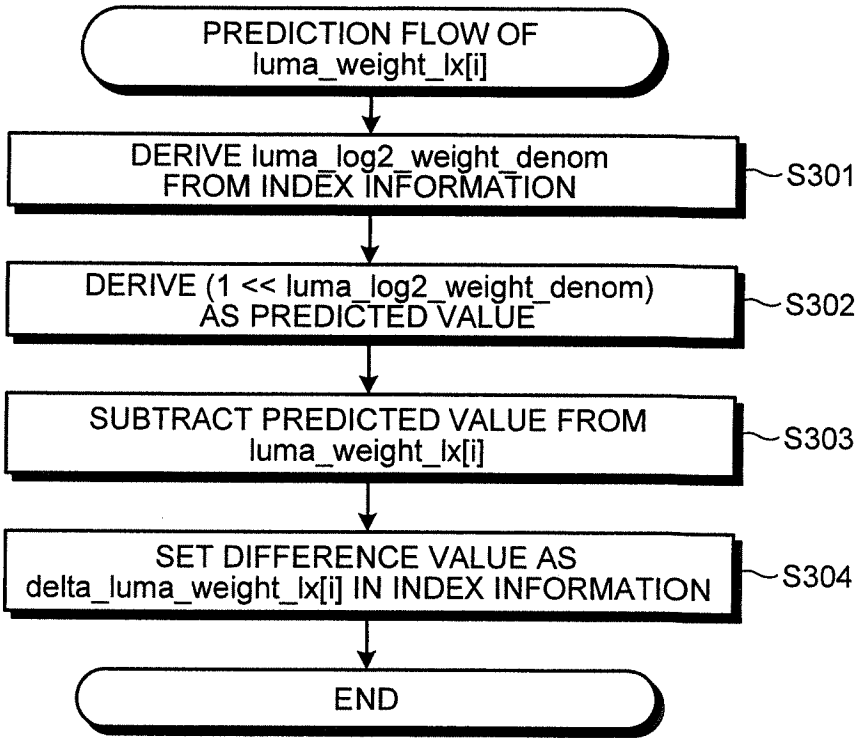

PREDICTION FLOW OF luma_weight_lx[i]
DERIVE luma_log2_weight_denom FROM INDEX INFORMATION
S301
DERIVE (1 << luma_log2_weight_denom) AS PREDICTED VALUE
S302
SUBTRACT PREDICTED VALUE FROM luma_weight_lx[i]
S303
SET DIFFERENCE VALUE AS delta_luma_weight_lx[i] IN INDEX INFORMATION
S304
END

FIG.17

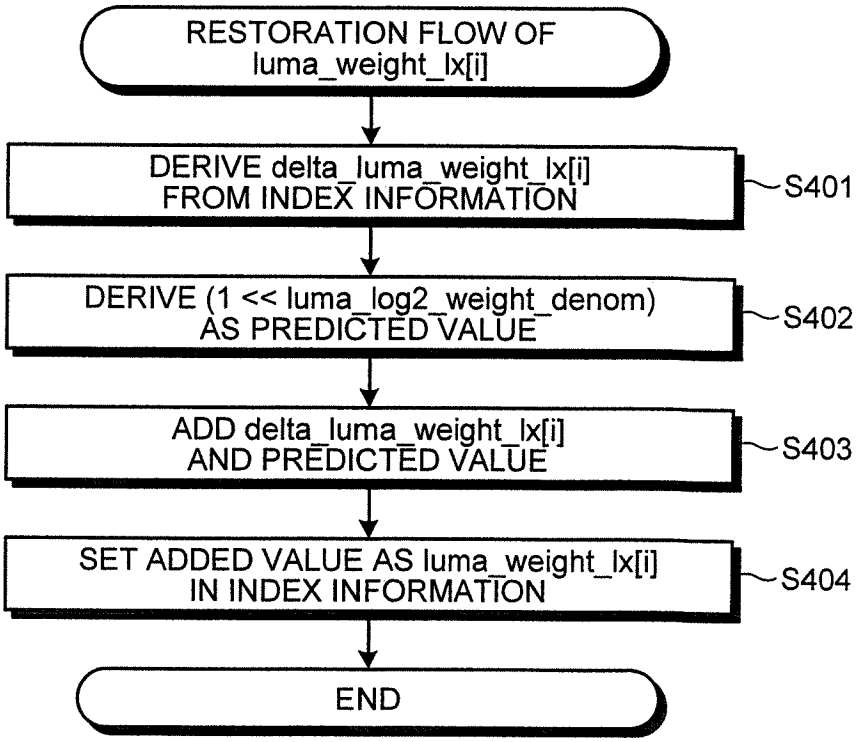

RESTORATION FLOW OF luma_weight_lx[i]
DERIVE delta_luma_weight_lx[i] FROM INDEX INFORMATION
S401
DERIVE (1 << luma_log2_weight_denom) AS PREDICTED VALUE
S402
ADD delta_luma_weight_lx[i] AND PREDICTED VALUE
S403
SET ADDED VALUE AS luma_weight_lx[i] IN INDEX INFORMATION
S404
END

FIG.18

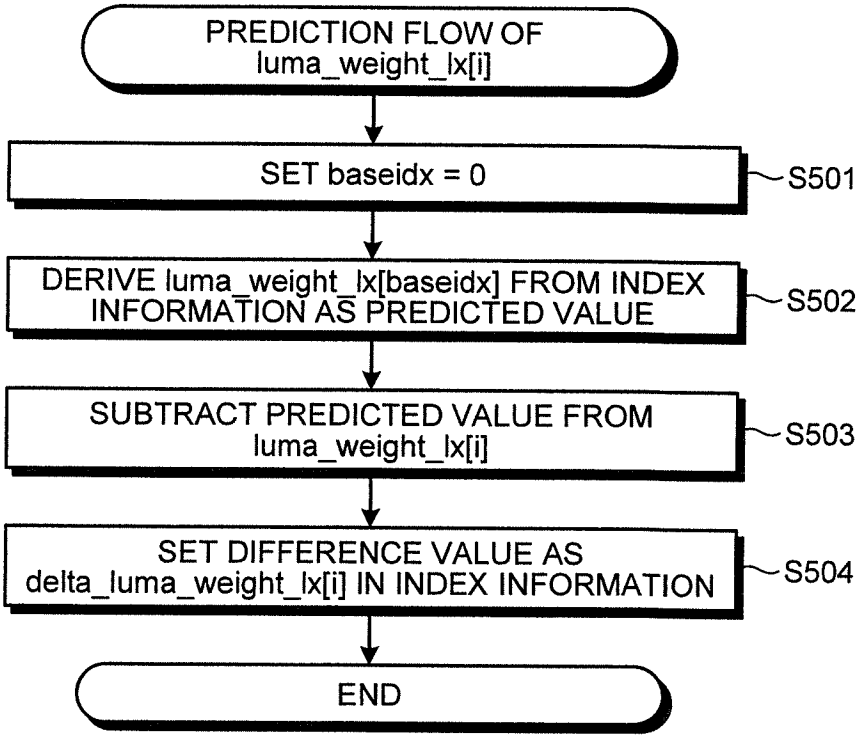

PREDICTION FLOW OF luma_weight_lx[i]
SET baseidx = 0
S501
DERIVE luma_weight_lx[baseidx] FROM INDEX INFORMATION AS PREDICTED VALUE
S502
SUBTRACT PREDICTED VALUE FROM luma_weight_lx[i]
S503
SET DIFFERENCE VALUE AS delta_luma_weight_lx[i] IN INDEX INFORMATION
S504
END

FIG.19

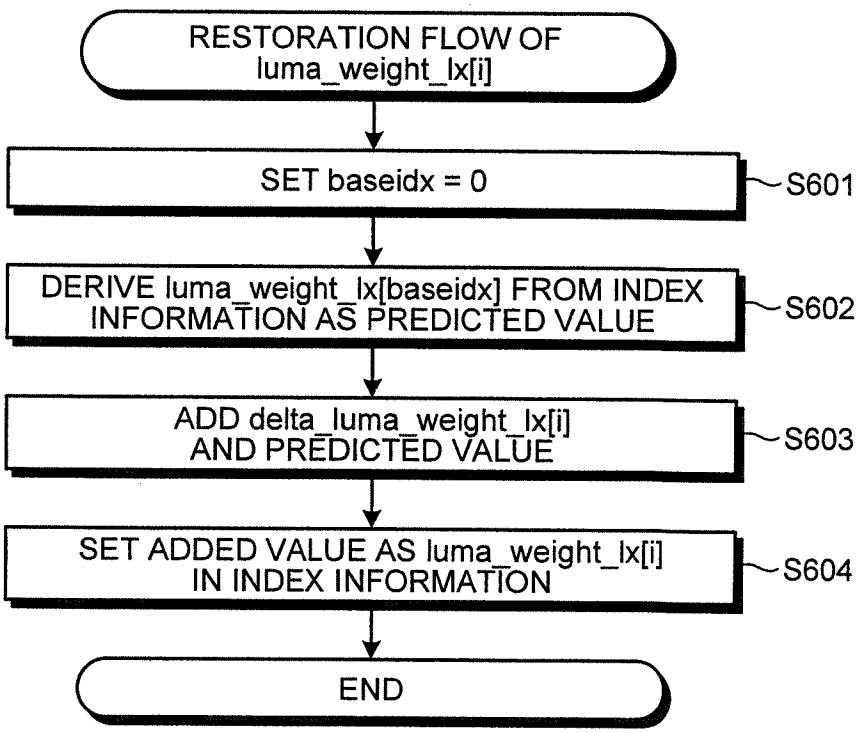

RESTORATION FLOW OF luma_weight_lx[i]
SET baseidx = 0
S601
DERIVE luma_weight_lx[baseidx] FROM INDEX INFORMATION AS PREDICTED VALUE
S602
ADD delta_luma_weight_lx[i] AND PREDICTED VALUE
S603
SET ADDED VALUE AS luma_weight_lx[i] IN INDEX INFORMATION
S604
END

FIG.20

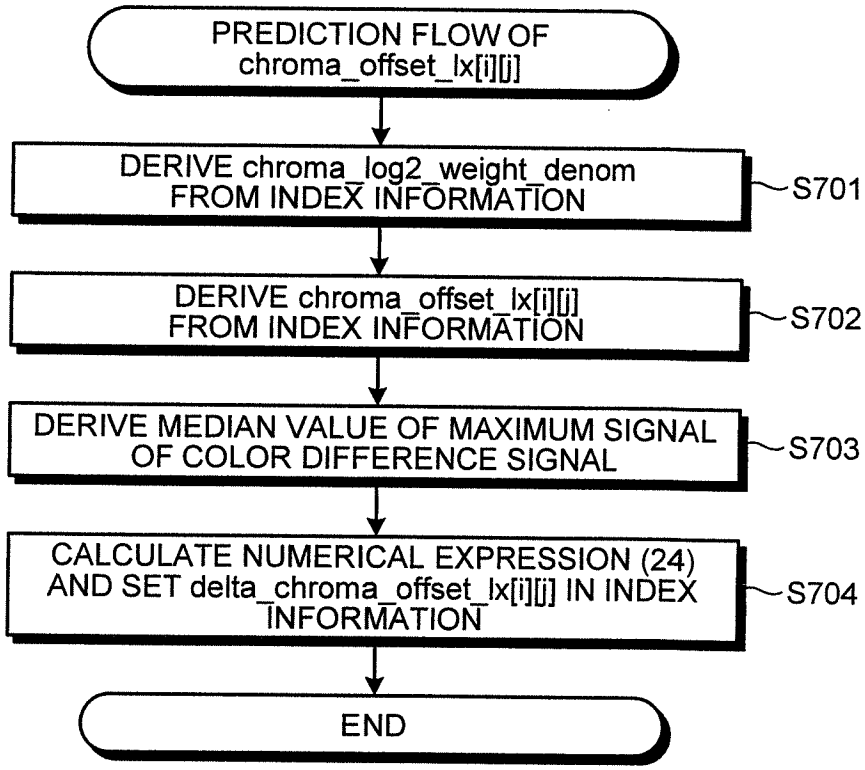

PREDICTION FLOW OF chroma_offset_lx[i][j]
DERIVE chroma_log2_weight_denom FROM INDEX INFORMATION
S701
DERIVE chroma_offset_lx[i][j] FROM INDEX INFORMATION
S702
DERIVE MEDIAN VALUE OF MAXIMUM SIGNAL OF COLOR DIFFERENCE SIGNAL
S703
CALCULATE NUMERICAL EXPRESSION (24) AND SET delta_chroma_offset_lx[i][j] IN INDEX INFORMATION
S704
END

FIG.21

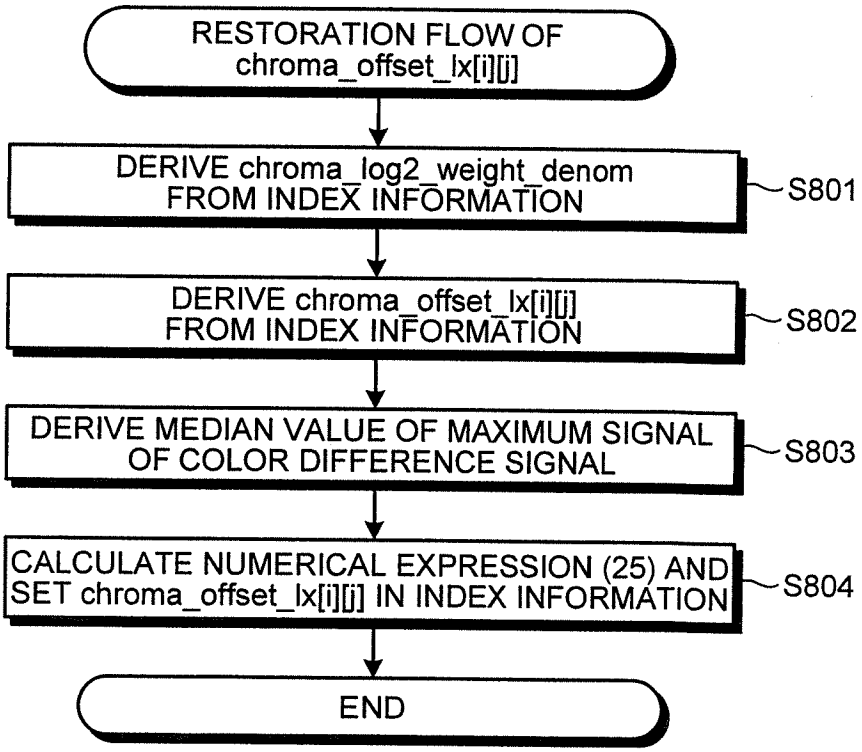

RESTORATION FLOW OF chroma_offset_lx[i][j]
DERIVE chroma_log2_weight_denom FROM INDEX INFORMATION
S801
DERIVE chroma_offset_lx[i][j] FROM INDEX INFORMATION
S802
DERIVE MEDIAN VALUE OF MAXIMUM SIGNAL OF COLOR DIFFERENCE SIGNAL
S803
CALCULATE NUMERICAL EXPRESSION (25) AND SET chroma_offset_lx[i][j] IN INDEX INFORMATION
S804
END

FIG.22

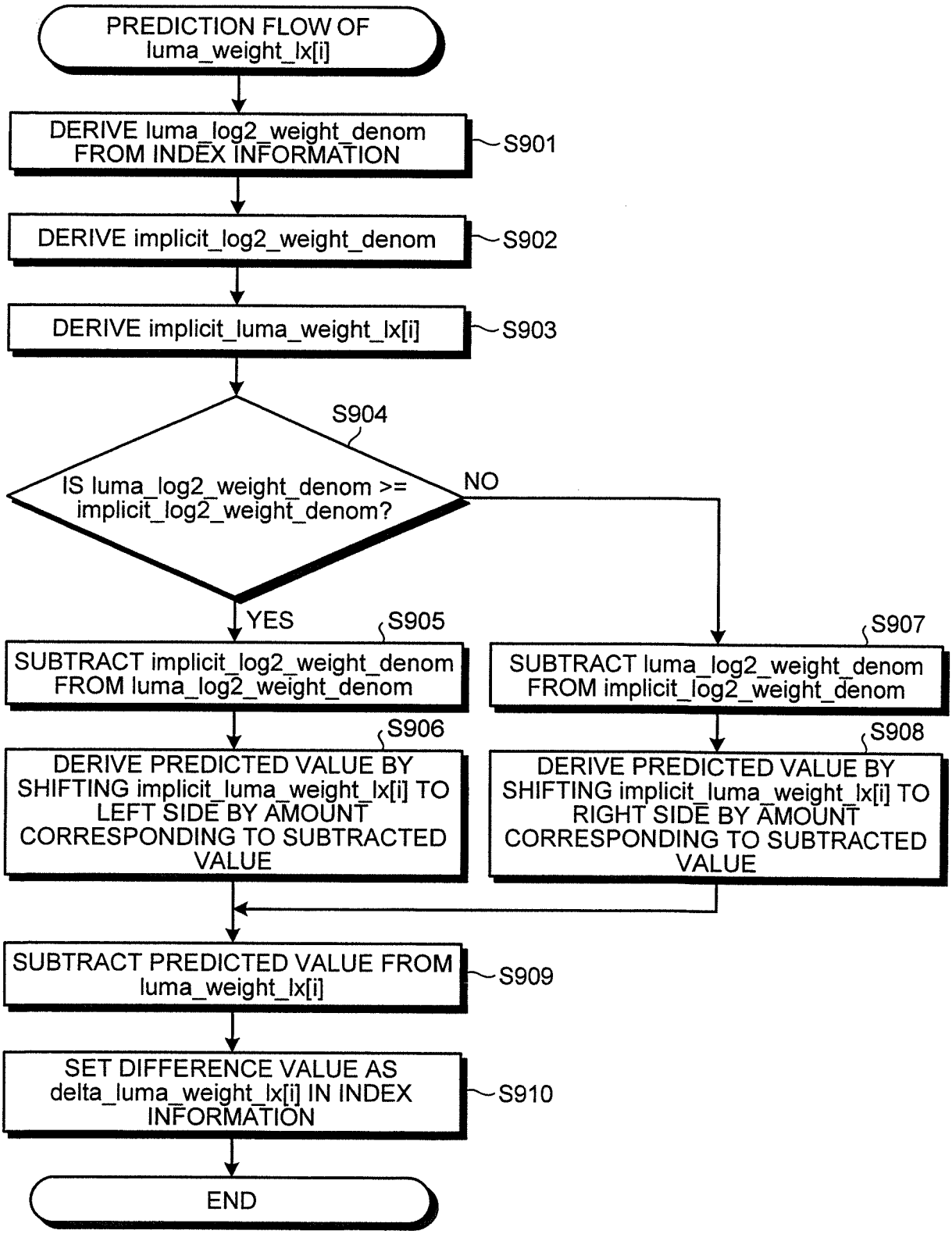

PREDICTION FLOW OF luma_weight_lx[i]
DERIVE luma_log2_weight_denom FROM INDEX INFORMATION
S901
DERIVE implicit_log2_weight_denom
S902
DERIVE implicit_luma_weight_lx[i]
S903
S904
IS luma_log2_weight_denom >= implicit_log2_weight_denom?
NO
YES
S905
SUBTRACT implicit_log2_weight_denom FROM luma_log2_weight_denom
S906
DERIVE PREDICTED VALUE BY SHIFTING implicit_luma_weight_lx[i] TO LEFT SIDE BY AMOUNT CORRESPONDING TO SUBTRACTED VALUE
S907
SUBTRACT luma_log2_weight_denom FROM implicit_log2_weight_denom
S908
DERIVE PREDICTED VALUE BY SHIFTING implicit_luma_weight_lx[i] TO RIGHT SIDE BY AMOUNT CORRESPONDING TO SUBTRACTED VALUE
SUBTRACT PREDICTED VALUE FROM luma_weight_lx[i]
S909
SET DIFFERENCE VALUE AS delta_luma_weight_lx[i] IN INDEX INFORMATION
S910
END

FIG.23

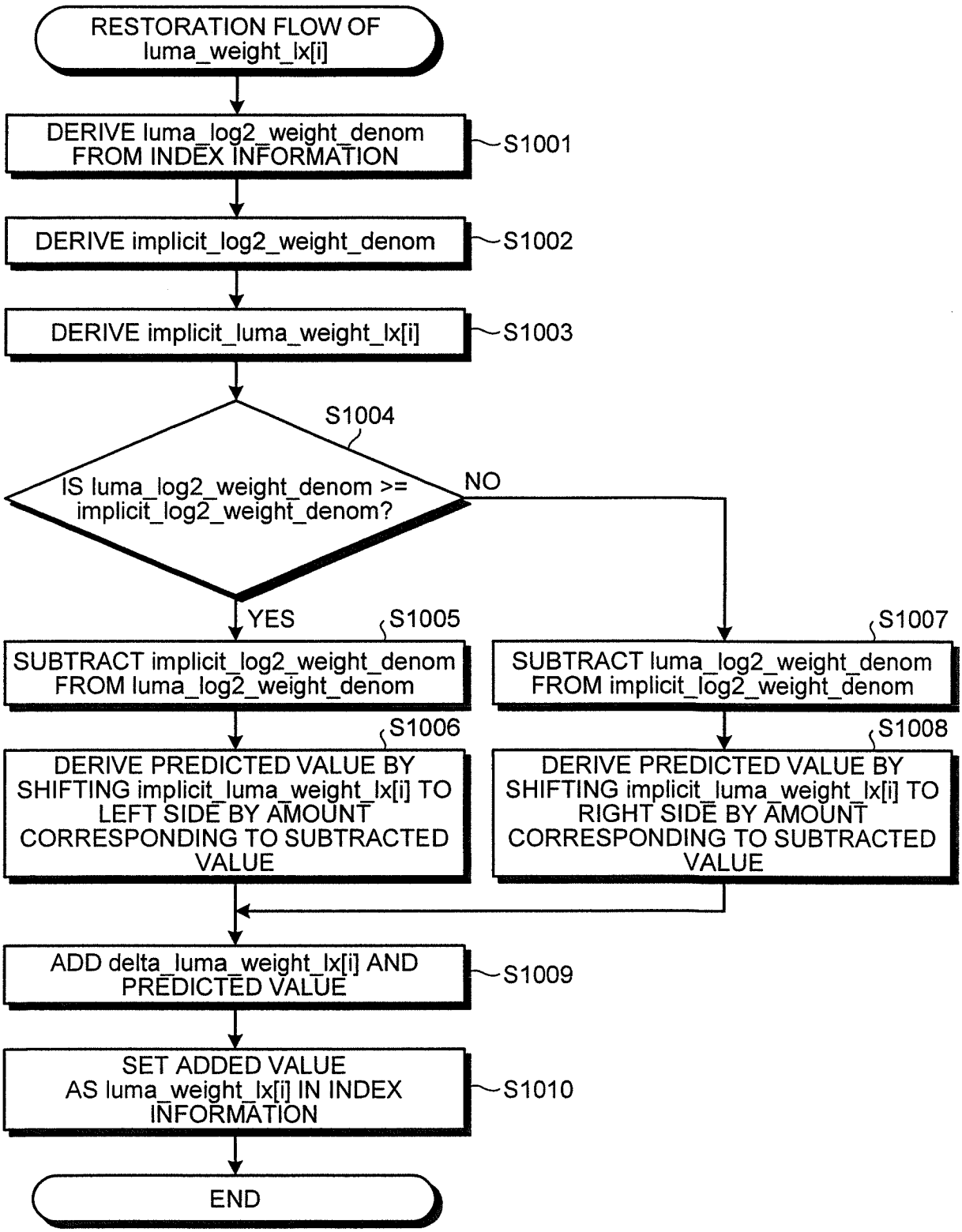
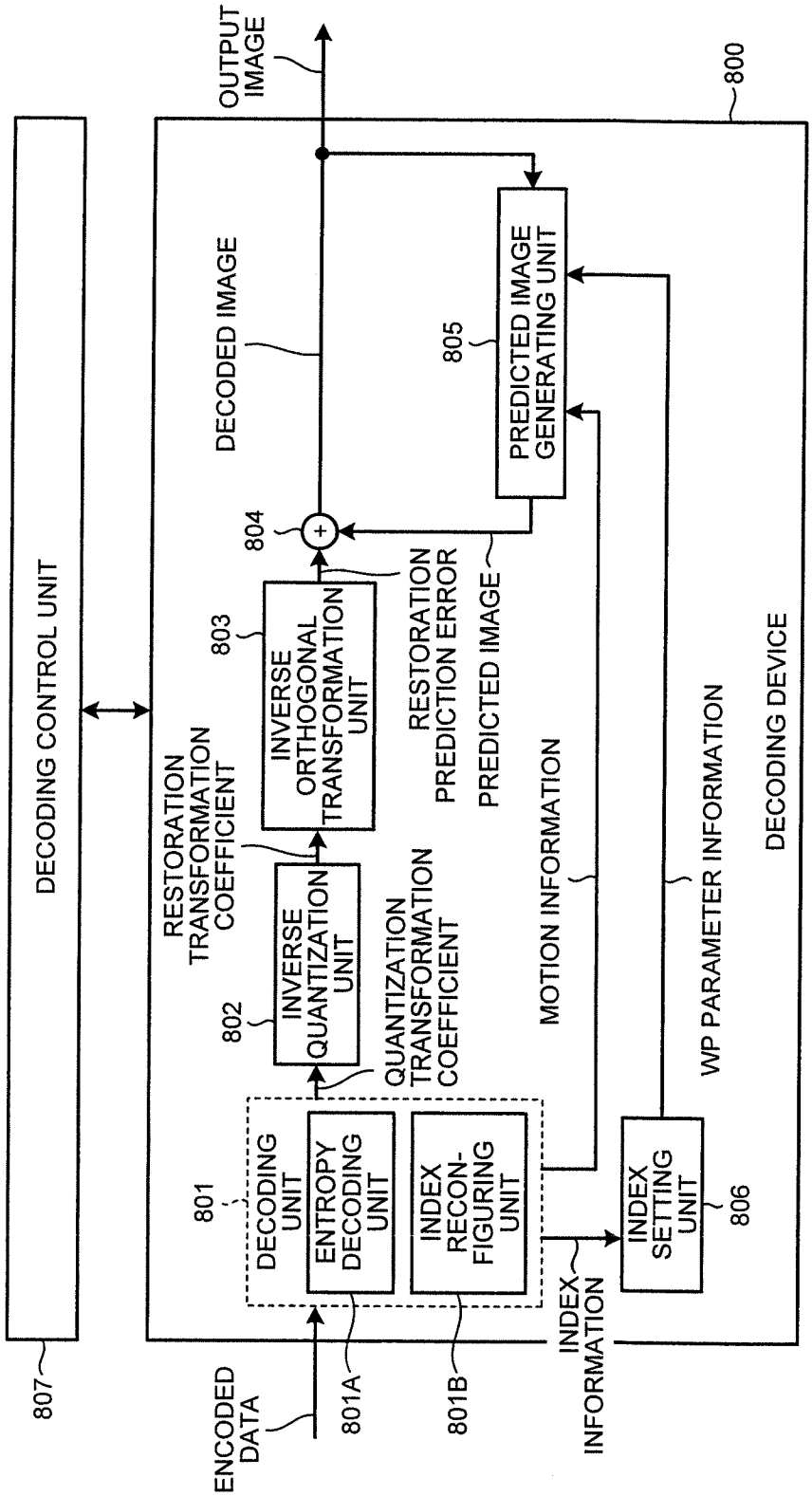

RESTORATION FLOW OF luma_weight_lx[i]
DERIVE luma_log2_weight_denom FROM INDEX INFORMATION
S1001
DERIVE implicit_log2_weight_denom
S1002
DERIVE implicit_luma_weight_lx[i]
S1003
S1004
IS luma_log2_weight_denom >= implicit_log2_weight_denom?
NO
YES
S1005
SUBTRACT implicit_log2_weight_denom FROM luma_log2_weight_denom
S1006
DERIVE PREDICTED VALUE BY SHIFTING implicit_luma_weight_lx[i] TO LEFT SIDE BY AMOUNT CORRESPONDING TO SUBTRACTED VALUE
S1007
SUBTRACT luma_log2_weight_denom FROM implicit_log2_weight_denom
S1008
DERIVE PREDICTED VALUE BY SHIFTING implicit_luma_weight_lx[i] TO RIGHT SIDE BY AMOUNT CORRESPONDING TO SUBTRACTED VALUE
ADD delta_luma_weight_lx[i] AND PREDICTED VALUE
S1009
SET ADDED VALUE AS luma_weight_lx[i] IN INDEX INFORMATION
S1010
END

FIG.24


DECODING CONTROL UNIT
807
ENCODED DATA
801
DECODING UNIT
ENTROPY DECODING UNIT
801A
INDEX RECON-FIGURING UNIT
801B
INDEX INFORMATION
802
INVERSE QUANTIZATION UNIT
QUANTIZATION TRANSFORMATION COEFFICIENT
RESTORATION TRANSFORMATION COEFFICIENT
803
INVERSE ORTHOGONAL TRANSFORMATION UNIT
804
RESTORATION PREDICTION ERROR
PREDICTED IMAGE
DECODED IMAGE
OUTPUT IMAGE
805
PREDICTED IMAGE GENERATING UNIT
MOTION INFORMATION
INDEX SETTING UNIT
806
WP PARAMETER INFORMATION
800
DECODING DEVICE