FIG.1

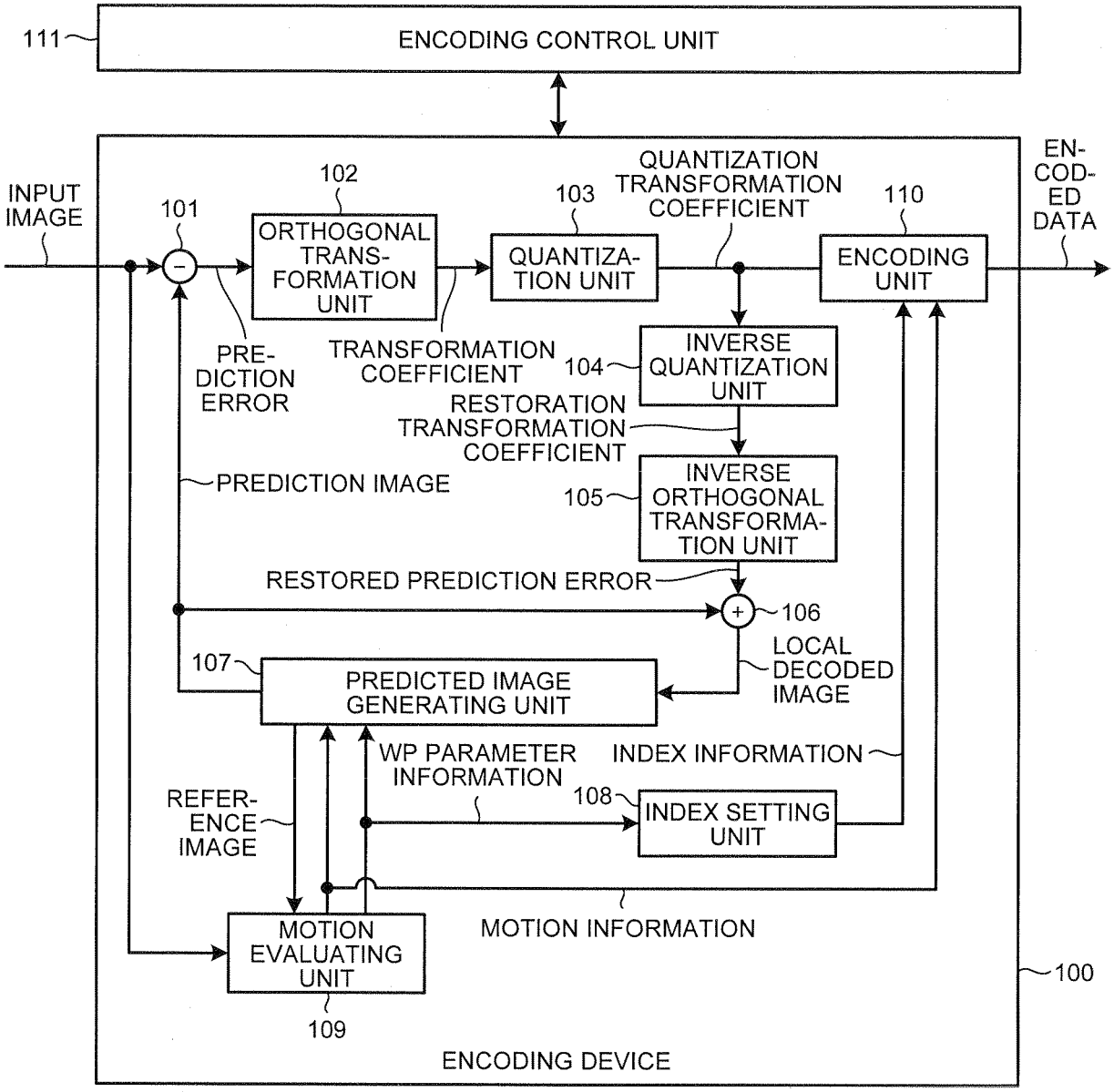

111
ENCODING CONTROL UNIT
INPUT IMAGE
101
102
ORTHOGONAL TRANS-FORMATION UNIT
103
QUANTIZA-TION UNIT
QUANTIZATION TRANSFORMATION COEFFICIENT
110
ENCODING UNIT
EN-COD-ED DATA
PRE-DICTION ERROR
TRANSFORMATION COEFFICIENT
104
INVERSE QUANTIZATION UNIT
RESTORATION TRANSFORMATION COEFFICIENT
PREDICTION IMAGE
105
INVERSE ORTHOGONAL TRANSFORMA-TION UNIT
RESTORED PREDICTION ERROR
106
LOCAL DECODED IMAGE
107
PREDICTED IMAGE GENERATING UNIT
WP PARAMETER INFORMATION
INDEX INFORMATION
REFER-ENCE IMAGE
108
INDEX SETTING UNIT
MOTION INFORMATION
MOTION EVALUATING UNIT
109
100
ENCODING DEVICE

FIG.2

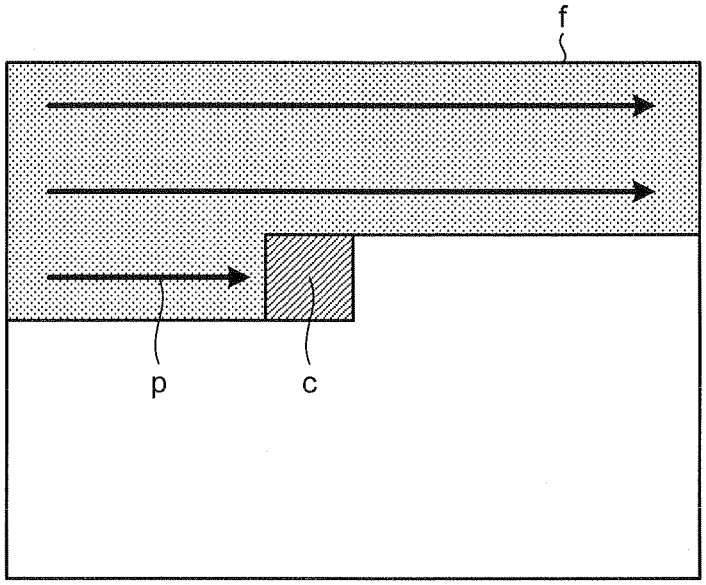


FIG.3A

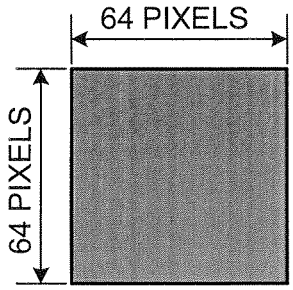


FIG.3B

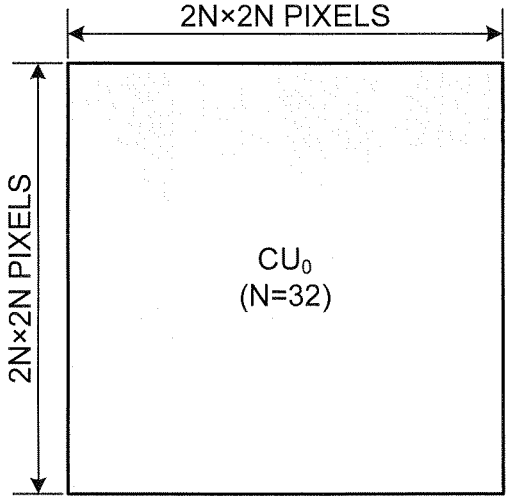


FIG.3C

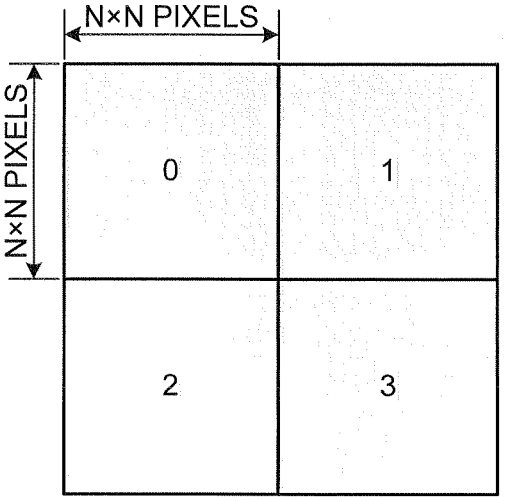


FIG.3D

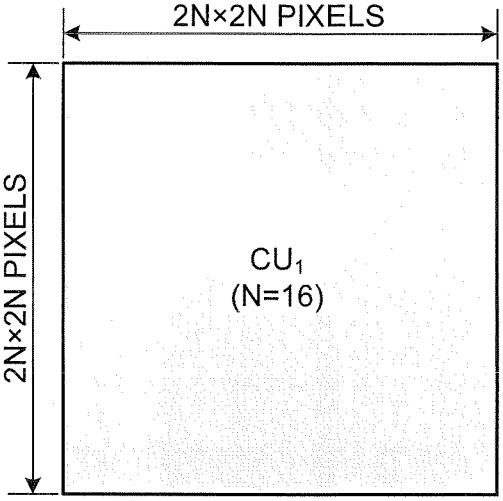


FIG.4

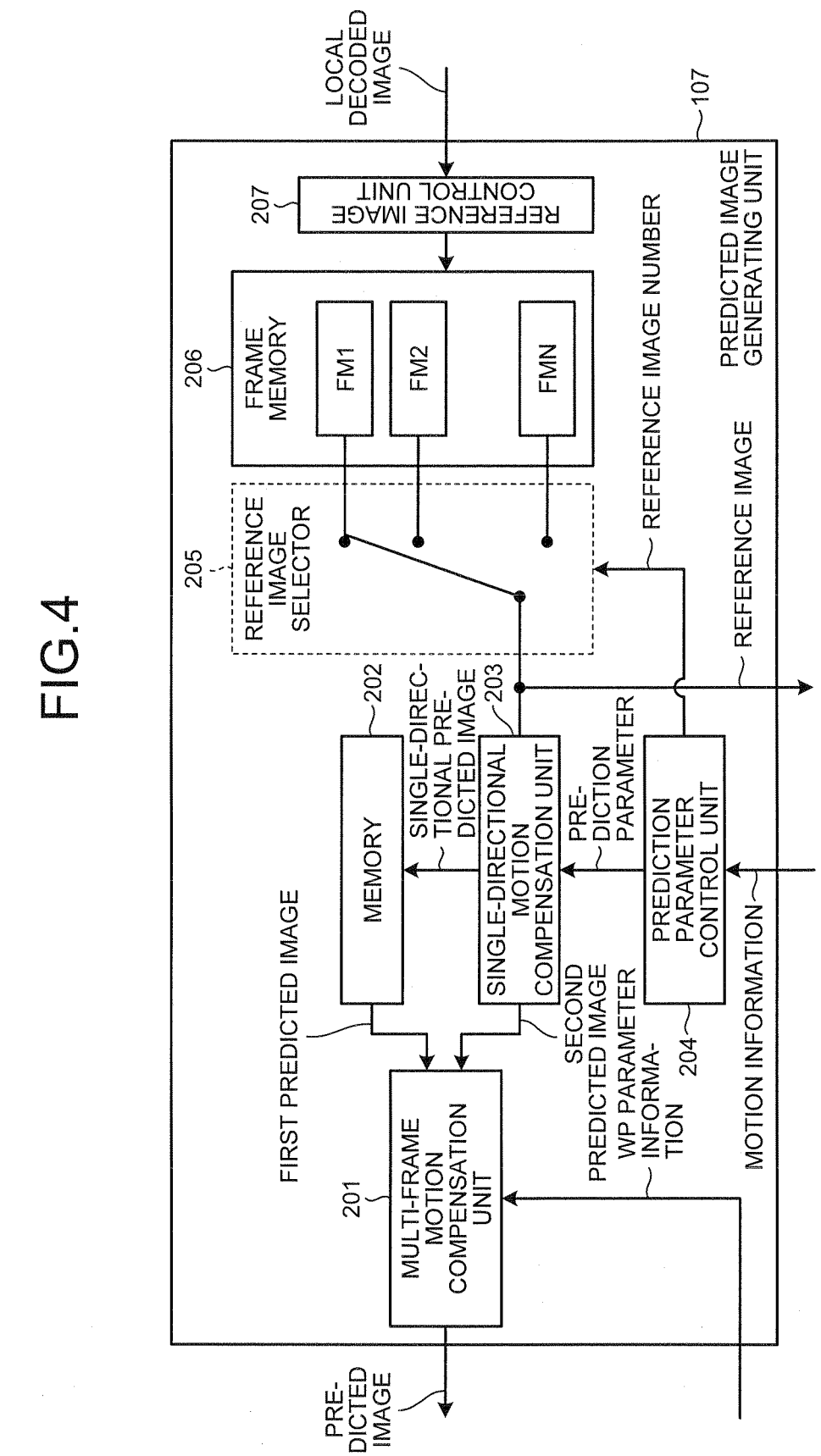

LOCAL DECODED IMAGE
107
PREDICTED IMAGE GENERATING UNIT
207
REFERENCE IMAGE CONTROL UNIT
206
FRAME MEMORY
FM1
FM2
FMN
205
REFERENCE IMAGE SELECTOR
REFERENCE IMAGE NUMBER
REFERENCE IMAGE
202
MEMORY
SINGLE-DIRECTIONAL PREDICTED IMAGE
203
SINGLE-DIRECTIONAL MOTION COMPENSATION UNIT
PREDICTION PARAMETER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

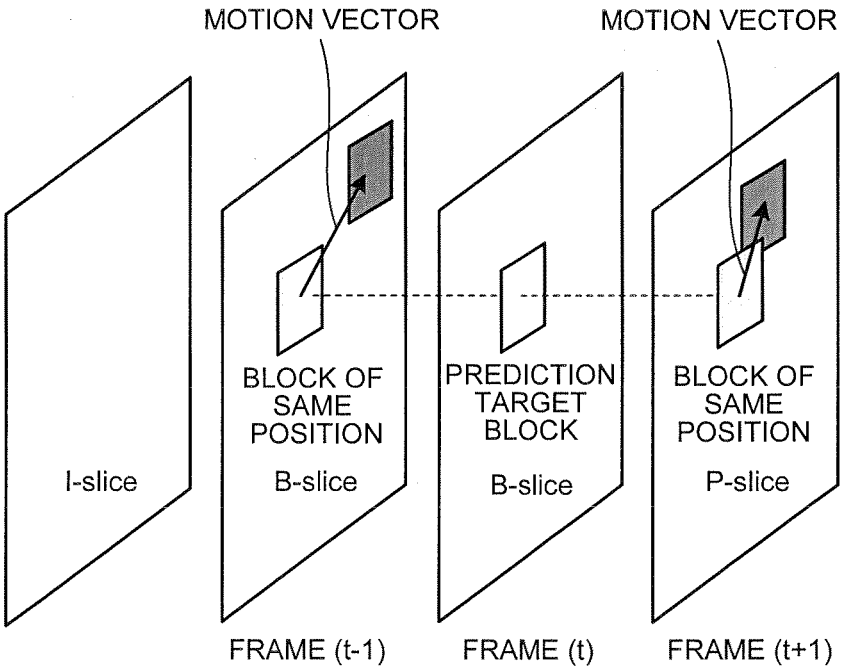

MOTION VECTOR
MOTION VECTOR
BLOCK OF SAME POSITION
PREDICTION TARGET BLOCK
BLOCK OF SAME POSITION
I-slice
B-slice
B-slice
P-slice
FRAME (t-1)
FRAME (t)
FRAME (t+1)

FIG.6

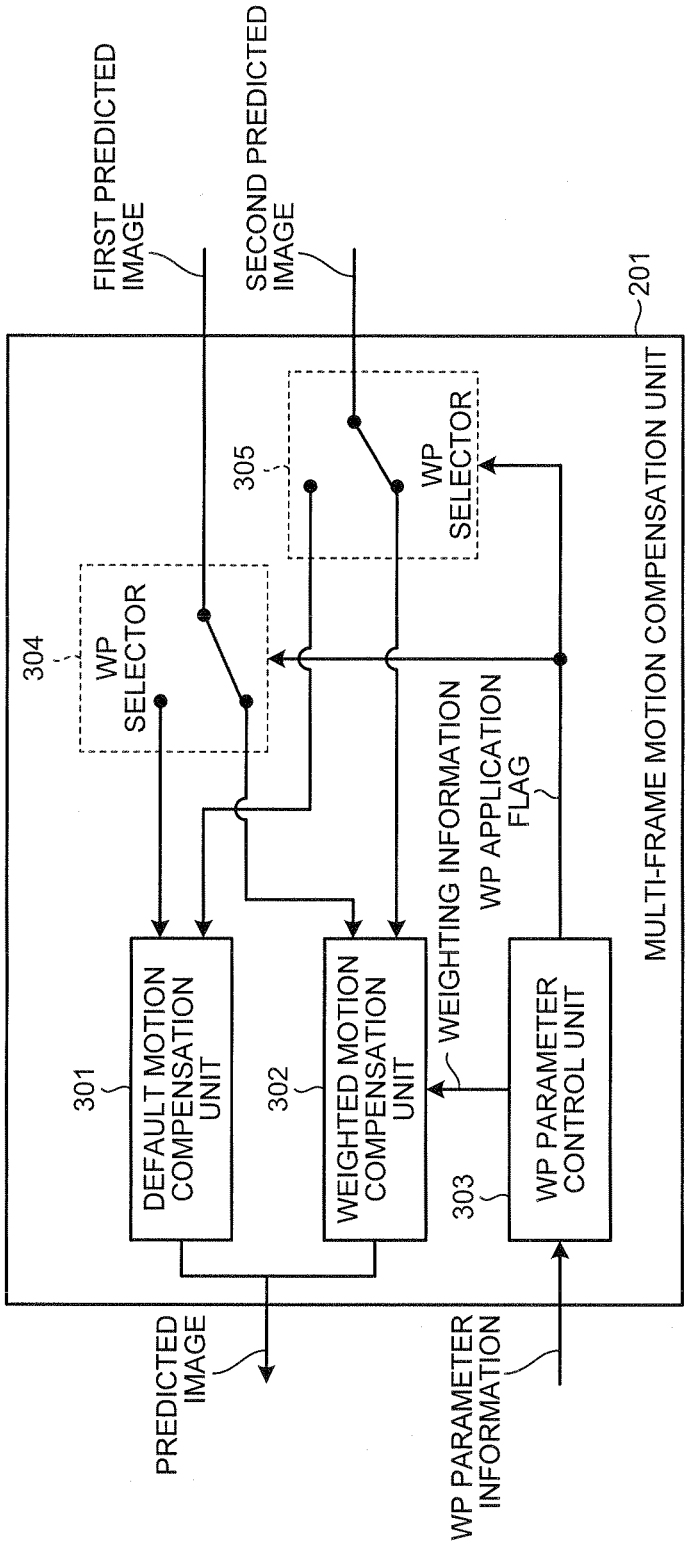

FIRST PREDICTED IMAGE
SECOND PREDICTED IMAGE
201
MULTI-FRAME MOTION COMPENSATION UNIT
304
WP SELECTOR
305
WP SELECTOR
301
DEFAULT MOTION COMPENSATION UNIT
302
WEIGHTED MOTION COMPENSATION UNIT
WEIGHTING INFORMATION
WP APPLICATION FLAG
303
WP PARAMETER CONTROL UNIT
PREDICTED IMAGE
WP PARAMETER INFORMATION

FIG.7

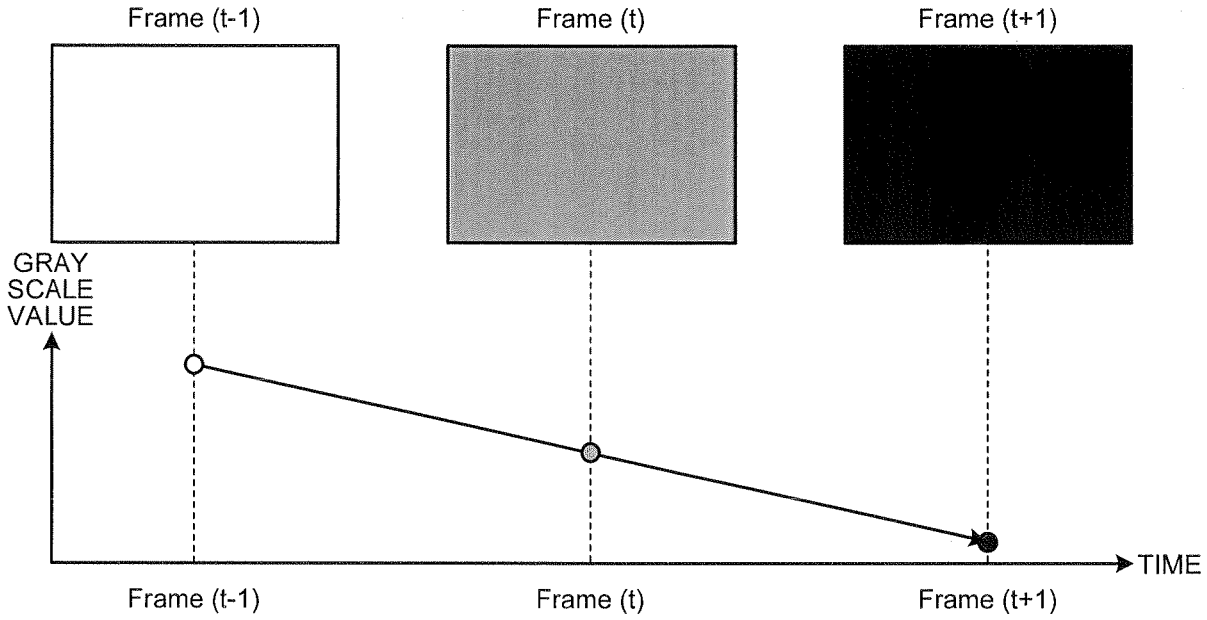

Frame (t-1)
Frame (t)
Frame (t+1)
GRAY SCALE VALUE
TIME
Frame (t-1)
Frame (t)
Frame (t+1)

FIG.8

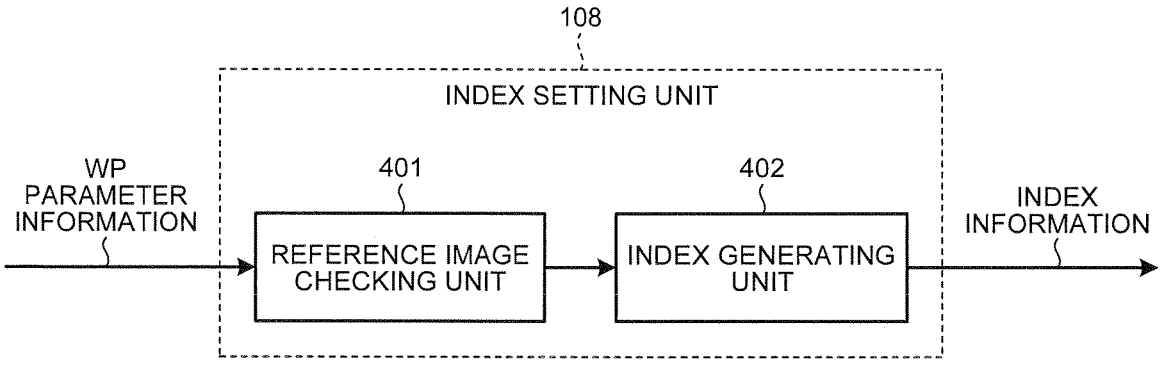

108
INDEX SETTING UNIT
WP PARAMETER INFORMATION
401
REFERENCE IMAGE CHECKING UNIT
402
INDEX GENERATING UNIT
INDEX INFORMATION

FIG.9A

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.9B

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.10

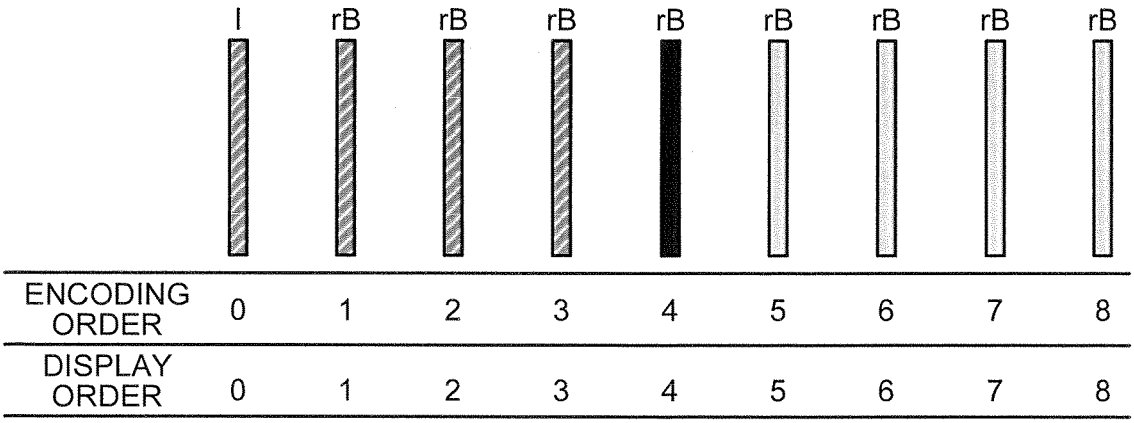
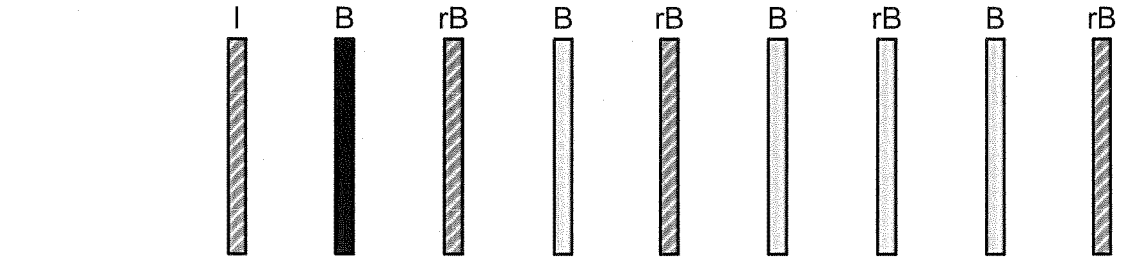


FIG.11

	REFERENCE NUMBER (refIdx)			
	0	1	2	3
LIST 0	3	2	1	0
LIST 1	3	2	1	0

FIG.12



ENCODING ORDER	0	4	3	5	2	7	6	8	1
DISPLAY ORDER	0	1	2	3	4	5	6	7	8

FIG.13

	REFERENCE NUMBER (refIdx)			
	0	1	2	3
LIST 0	0	2	4	8
LIST 1	2	4	8	0

FIG.14

SCANNING ORDER	REFERENCE NUMBER (refIdx)					
	0	1	2	3	...	N
LIST 0	0	2	4	6		2N
LIST 1	1	3	5	7		2N+1

FIG.15

	REFERENCE NUMBER (refIdx)			
	0	1	2	3
COMMON LIST	3	2	1	0

FIG.16

	REFERENCE NUMBER (refIdx)			
	0	1	2	3
COMMON LIST	0	2	4	8

FIG.17

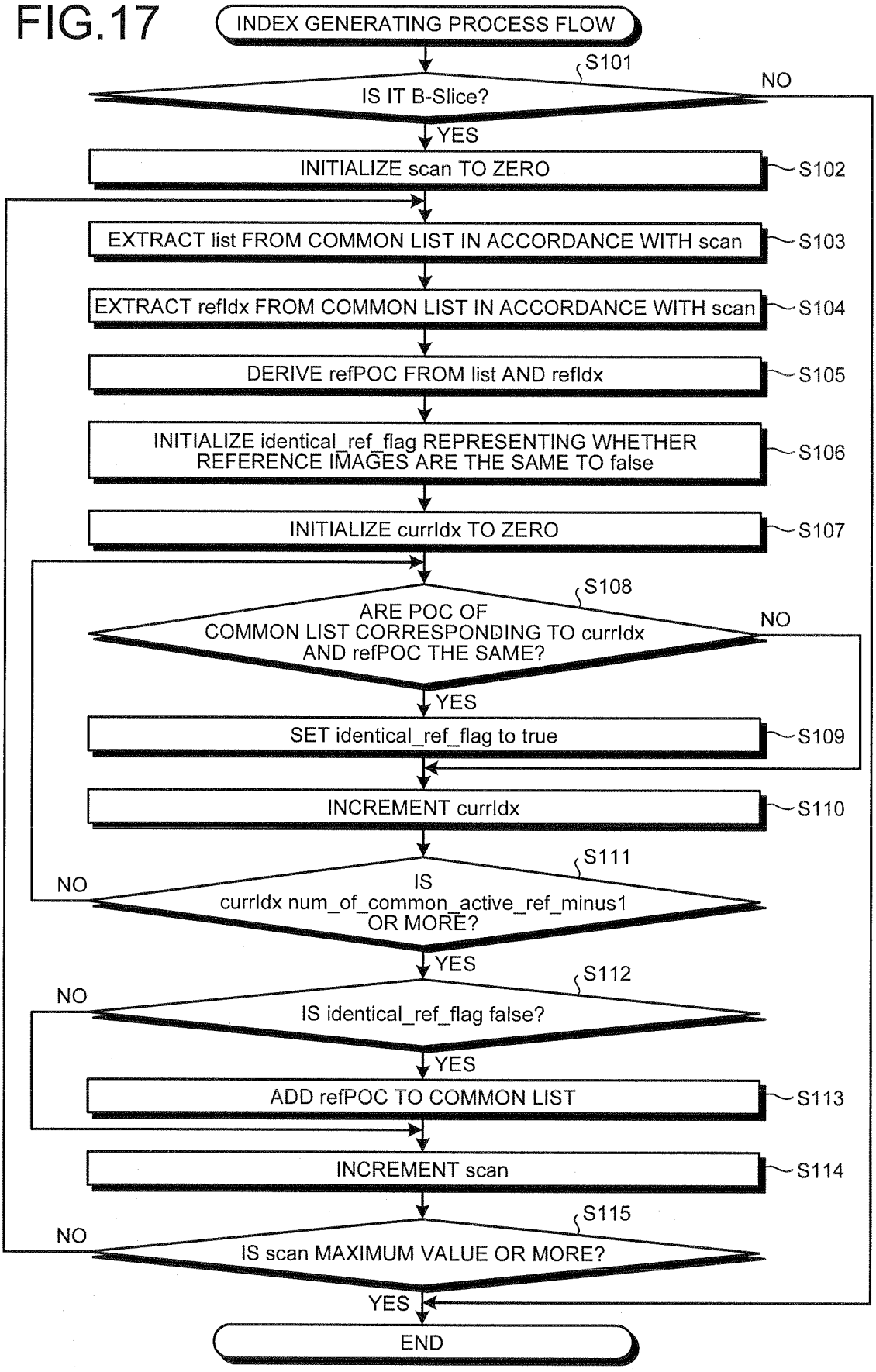

INDEX GENERATING PROCESS FLOW
S101
IS IT B-Slice?
NO
YES
INITIALIZE scan TO ZERO
S102
EXTRACT list FROM COMMON LIST IN ACCORDANCE WITH scan
S103
EXTRACT refIdx FROM COMMON LIST IN ACCORDANCE WITH scan
S104
DERIVE refPOC FROM list AND refIdx
S105
INITIALIZE identical_ref_flag REPRESENTING WHETHER REFERENCE IMAGES ARE THE SAME TO false
S106
INITIALIZE currIdx TO ZERO
S107
S108
ARE POC OF COMMON LIST CORRESPONDING TO currIdx AND refPOC THE SAME?
NO
YES
SET identical_ref_flag to true
S109
INCREMENT currIdx
S110
S111
NO
IS currIdx num_of_common_active_ref_minus1 OR MORE?
YES
S112
NO
IS identical_ref_flag false?
YES
ADD refPOC TO COMMON LIST
S113
INCREMENT scan
S114
S115
NO
IS scan MAXIMUM VALUE OR MORE?
YES
END

FIG.18

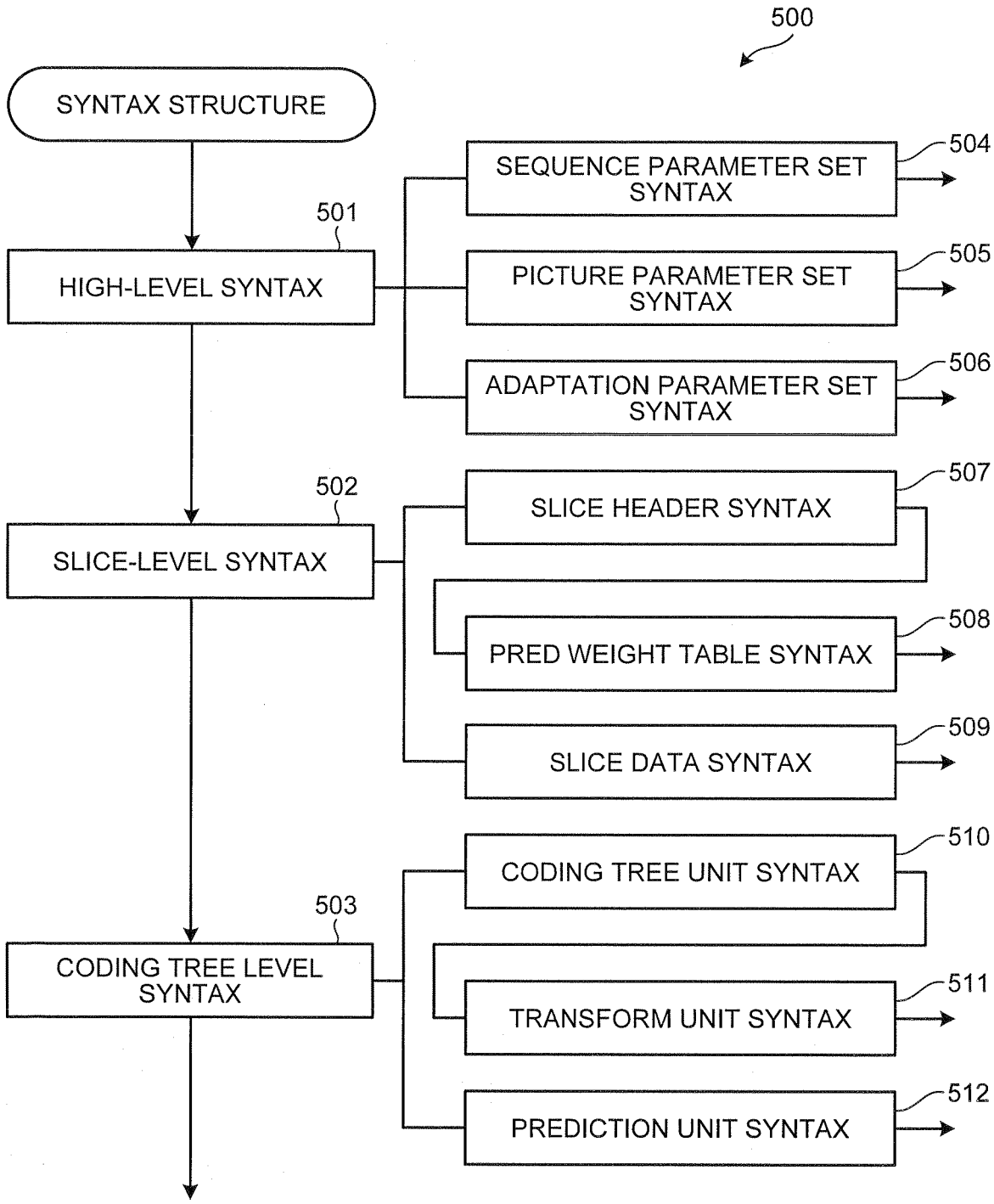

500
SYNTAX STRUCTURE
501
HIGH-LEVEL SYNTAX
502
SLICE-LEVEL SYNTAX
503
CODING TREE LEVEL SYNTAX
SEQUENCE PARAMETER SET SYNTAX
504
PICTURE PARAMETER SET SYNTAX
505
ADAPTATION PARAMETER SET SYNTAX
506
SLICE HEADER SYNTAX
507
PRED WEIGHT TABLE SYNTAX
508
SLICE DATA SYNTAX
509
CODING TREE UNIT SYNTAX
510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

FIG.19

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.20

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.21

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_common_active_minus1; i++) {	
luma_weight_common_flag	u(1)
if(luma_weight_common_flag) {	
luma_weight_common[i]	se(v)
luma_offset_common[i]	se(v)
}	
if (chroma_format_idc != MONO_IDX) {	
chroma_weight_common_flag	u(1)
if(chroma_weight_common_flag)	
for(j =0; j < COLOR_COMP − 1;j++) {	
chroma_weight_common[i][j]	se(v)
chroma_offset_common[i][j]	se(v)
}	
}	
}	

FIG.22

seq_parameter_set_rbsp( ) {	**Descriptor**
profile_idc	u(8)
reserved_zero_8bits /* equal to 0 */	u(8)
level_idc	u(8)
seq_parameter_set_id	ue(v)
…	
num_ref_frames	ue(v)
…	
weighted_prediction_enabled_flag	u(1)
…	
rbsp_trailing_bits( )	
}	

FIG.23

aps_rbsp( ) {	**Descriptor**
aps_id	ue(v)
...	
aps_weighted_prediction_flag	u(1)
aps_weighted_bipred_idx	u(2)
...	
if(aps_weighted_prediction_flag | | aps_weighted_bipred_idx = = 1)	
pred_weight_table( )	
...	
rbsp_trailing_bits( )	
}	

FIG.24

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 <= MAX_COMMON_REF_MINUS1; i++) {	
luma_weight_common_flag	u(1)
if(luma_weight_common_flag) {	
luma_weight_common[i]	se(v)
luma_offset_common[i]	se(v)
}	
if (chroma_format_idc != MONO_IDX) {	
chroma_weight_common_flag	u(1)
if(chroma_weight_common_flag)	
for(j =0; j < COLOR_COMP - 1; j++) {	
chroma_weight_common[i][j]	se(v)
chroma_offset_common[i][j]	se(v)
}	
}	
}	

FIG.25

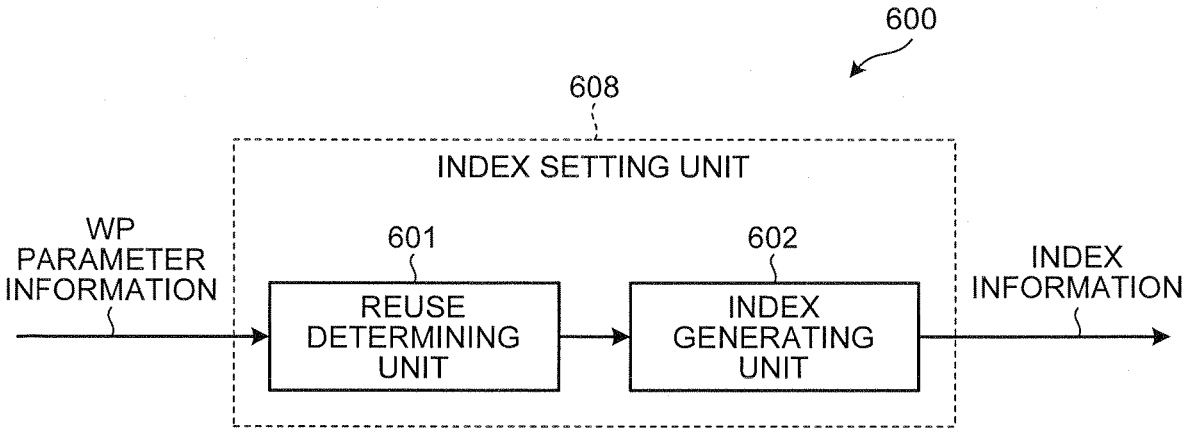

600
608
INDEX SETTING UNIT
WP
PARAMETER
INFORMATION
601
REUSE
DETERMINING
UNIT
602
INDEX
GENERATING
UNIT
INDEX
INFORMATION

FIG.26

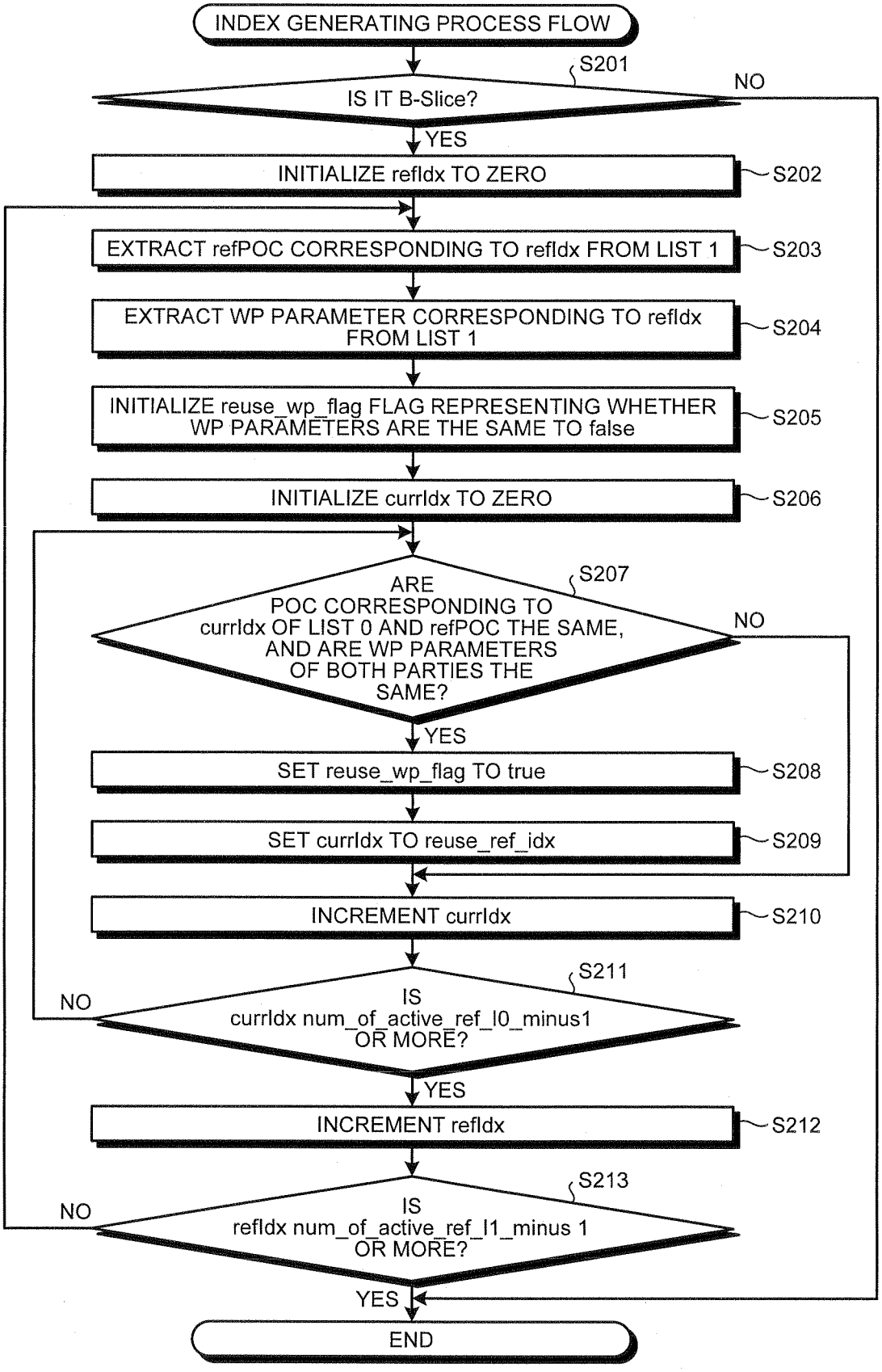

INDEX GENERATING PROCESS FLOW
S201
IS IT B-Slice?
NO
YES
INITIALIZE refIdx TO ZERO
S202
EXTRACT refPOC CORRESPONDING TO refIdx FROM LIST 1
S203
EXTRACT WP PARAMETER CORRESPONDING TO refIdx FROM LIST 1
S204
INITIALIZE reuse_wp_flag FLAG REPRESENTING WHETHER WP PARAMETERS ARE THE SAME TO false
S205
INITIALIZE currIdx TO ZERO
S206
S207
ARE POC CORRESPONDING TO currIdx OF LIST 0 AND refPOC THE SAME, AND ARE WP PARAMETERS OF BOTH PARTIES THE SAME?
NO
YES
SET reuse_wp_flag TO true
S208
SET currIdx TO reuse_ref_idx
S209
INCREMENT currIdx
S210
S211
IS currIdx num_of_active_ref_l0_minus1 OR MORE?
NO
YES
INCREMENT refIdx
S212
S213
IS refIdx num_of_active_ref_l1_minus 1 OR MORE?
NO
YES
END

FIG.27

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++) {	
reuse_wp_flag	u(1)
if(reuse_wp_flag) {	
reuse_ref_idx	se(v)
} else {	
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)

FIG.28

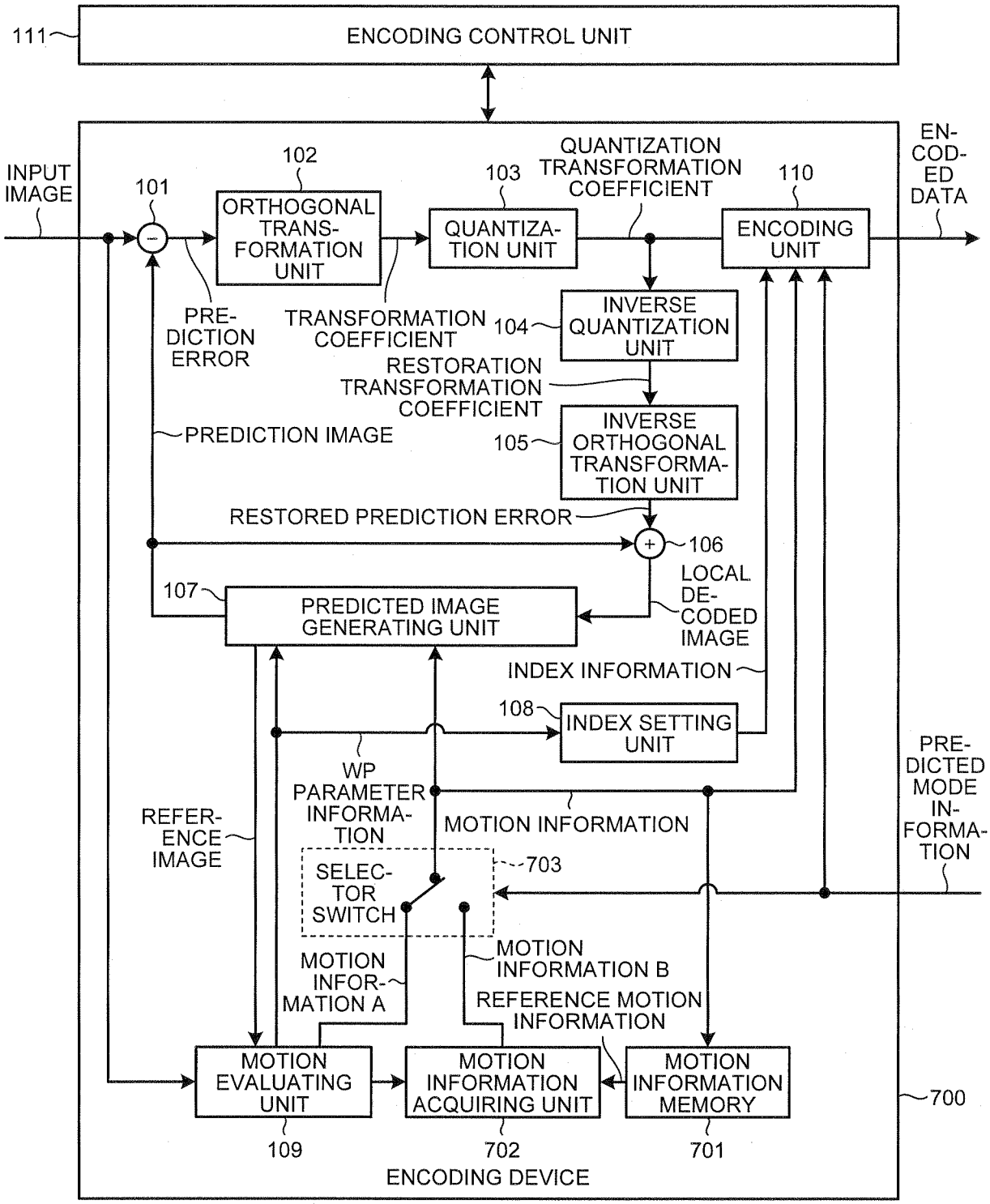

111
ENCODING CONTROL UNIT
INPUT IMAGE
101
102
ORTHOGONAL TRANS-FORMATION UNIT
103
QUANTIZA-TION UNIT
QUANTIZATION TRANSFORMATION COEFFICIENT
110
ENCODING UNIT
EN-COD-ED DATA
PRE-DICTION ERROR
TRANSFORMATION COEFFICIENT
104
INVERSE QUANTIZATION UNIT
RESTORATION TRANSFORMATION COEFFICIENT
PREDICTION IMAGE
105
INVERSE ORTHOGONAL TRANSFORMA-TION UNIT
RESTORED PREDICTION ERROR
106
LOCAL DE-CODED IMAGE
107
PREDICTED IMAGE GENERATING UNIT
INDEX INFORMATION
108
INDEX SETTING UNIT
WP PARAMETER INFORMA-TION
MOTION INFORMATION
PRE-DICTED MODE IN-FORMA-TION
REFER-ENCE IMAGE
SELEC-TOR SWITCH
703
MOTION INFOR-MATION A
MOTION INFORMATION B
REFERENCE MOTION INFORMATION
MOTION EVALUATING UNIT
109
MOTION INFORMATION ACQUIRING UNIT
702
MOTION INFORMATION MEMORY
701
700
ENCODING DEVICE

FIG.29

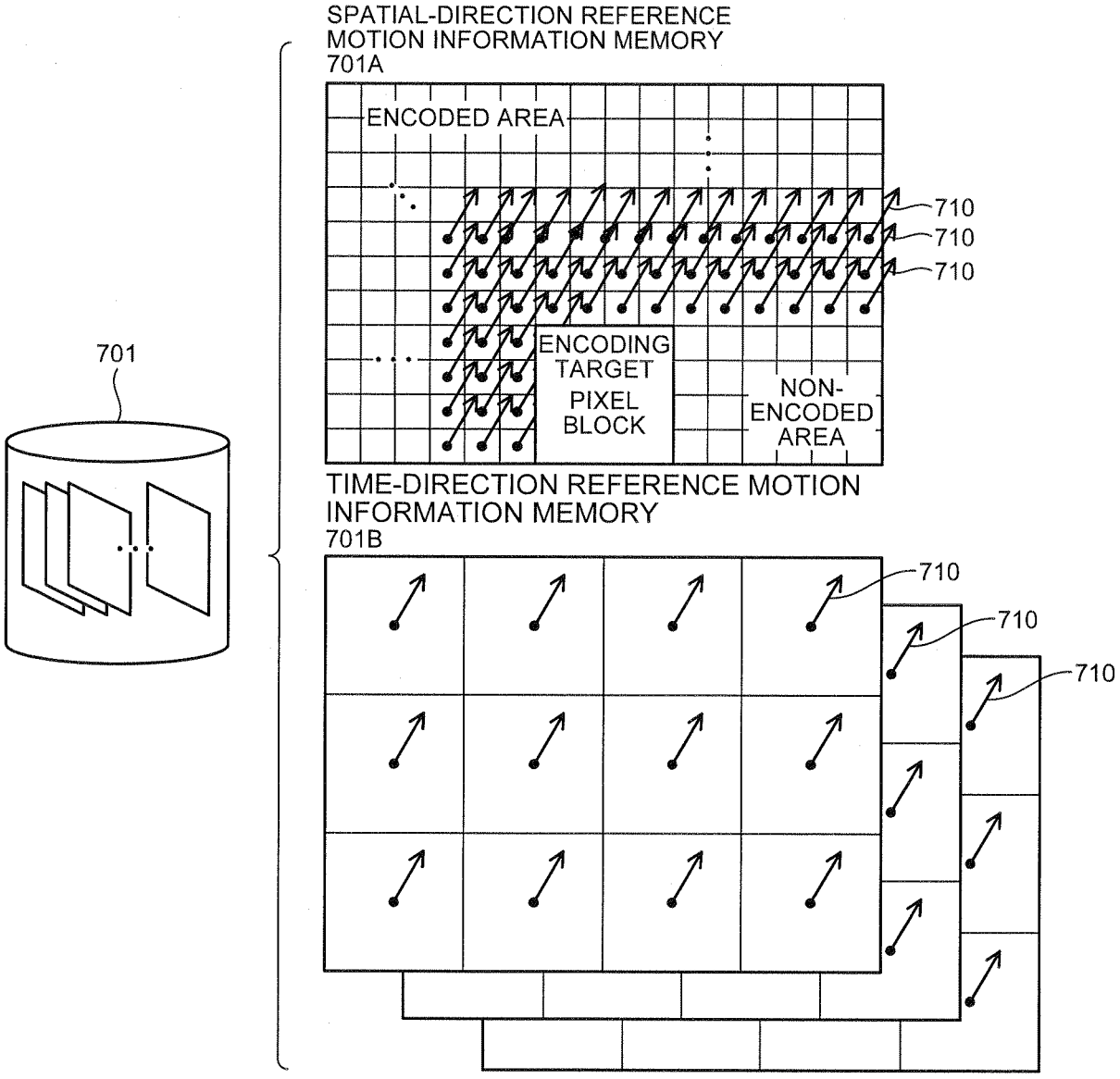

SPATIAL-DIRECTION REFERENCE MOTION INFORMATION MEMORY
701A
ENCODED AREA
710
710
710
ENCODING TARGET PIXEL BLOCK
NON-ENCODED AREA
701
TIME-DIRECTION REFERENCE MOTION INFORMATION MEMORY
701B
710
710
710

FIG.30A

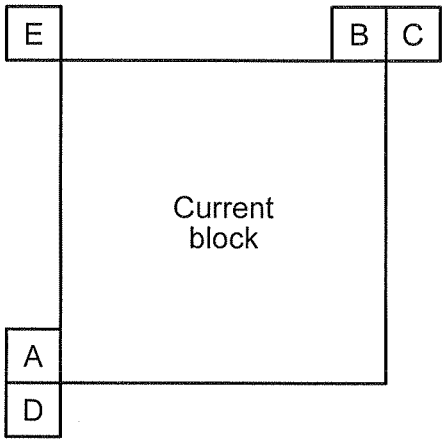


FIG.30B

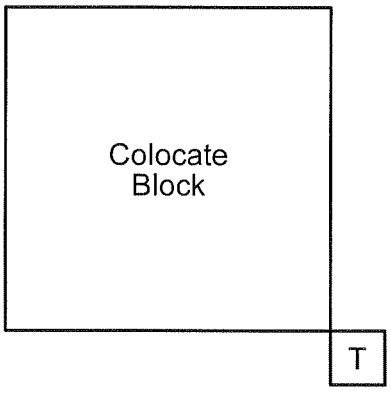


FIG.31

PIXEL BLOCK POSITION	Available?	PIXEL BLOCK POSITION INDEX (idx)
A	YES	0
B	YES	1
C	NO	-
D	YES	2
E	YES	3
T	YES	4

FIG.32

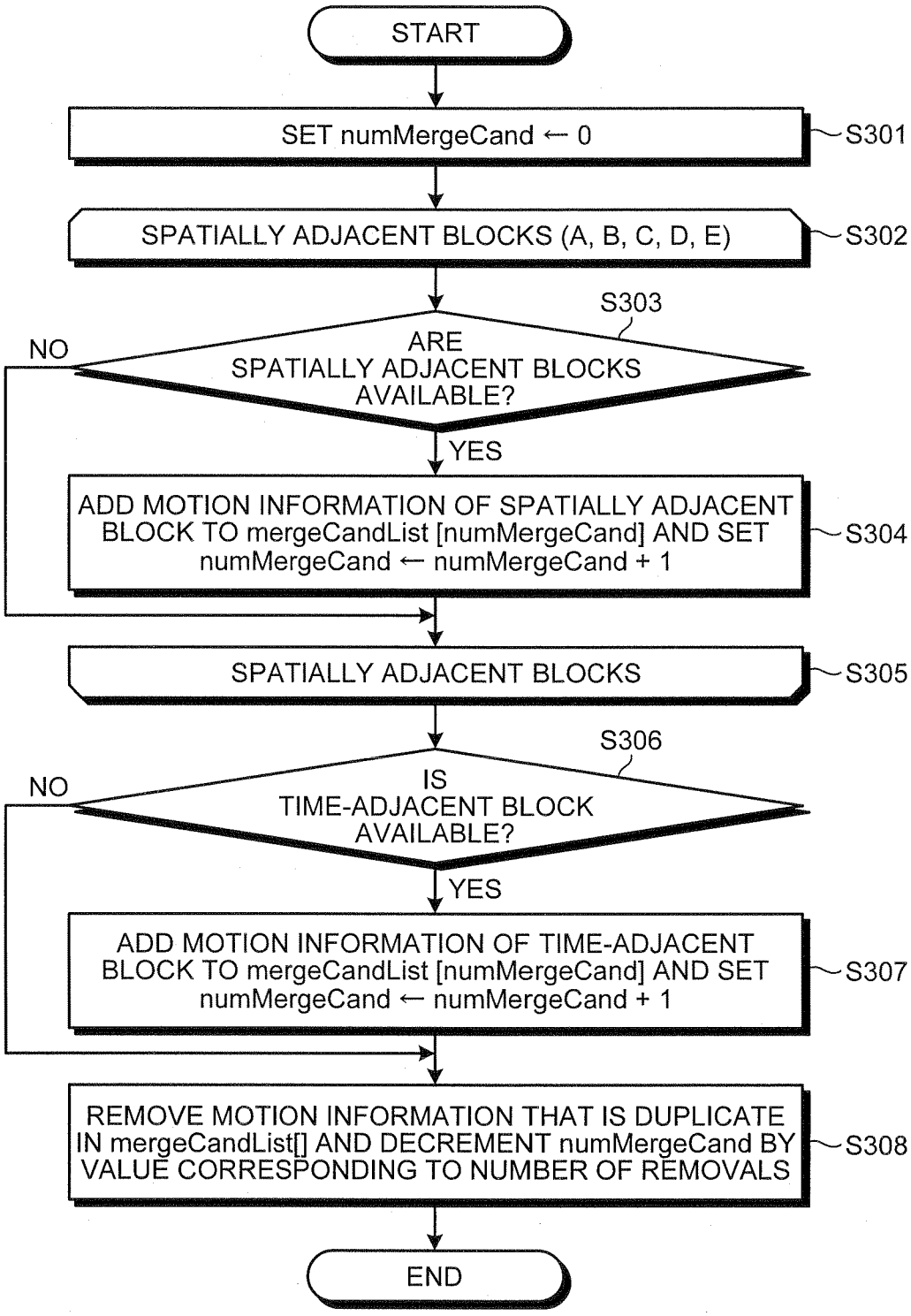

START
SET numMergeCand ← 0
S301
SPATIALLY ADJACENT BLOCKS (A, B, C, D, E)
S302
S303
ARE SPATIALLY ADJACENT BLOCKS AVAILABLE?
NO
YES
ADD MOTION INFORMATION OF SPATIALLY ADJACENT BLOCK TO mergeCandList [numMergeCand] AND SET numMergeCand ← numMergeCand + 1
S304
SPATIALLY ADJACENT BLOCKS
S305
S306
IS TIME-ADJACENT BLOCK AVAILABLE?
NO
YES
ADD MOTION INFORMATION OF TIME-ADJACENT BLOCK TO mergeCandList [numMergeCand] AND SET numMergeCand ← numMergeCand + 1
S307
REMOVE MOTION INFORMATION THAT IS DUPLICATE IN mergeCandList[] AND DECREMENT numMergeCand BY VALUE CORRESPONDING TO NUMBER OF REMOVALS
S308
END

FIG.33

idx	MergeCandList [idx]	
	REFERENCE LIST 0	REFERENCE LIST 1
0	refIdx0, mv0	--
1	--	refIdx1, mv1
2	refIdx2, mv2	--
3	refIdx3, mv3	refIdx3, mv3
4	refIdx4, mv4	refIdx4, mv4

FIG.34

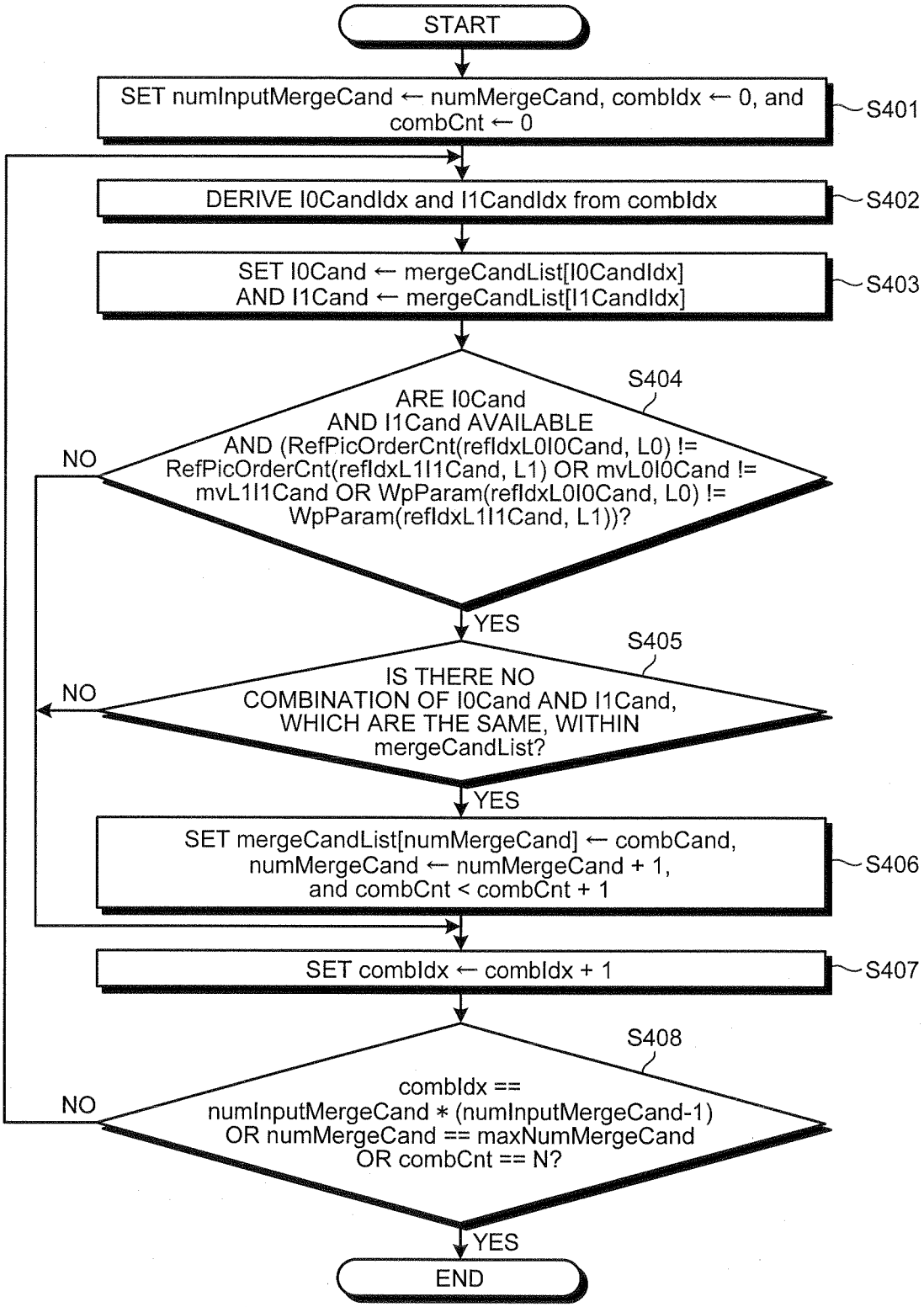

START
SET numInputMergeCand ← numMergeCand, combIdx ← 0, and combCnt ← 0
S401
DERIVE l0CandIdx and l1CandIdx from combIdx
S402
SET l0Cand ← mergeCandList[l0CandIdx] AND l1Cand ← mergeCandList[l1CandIdx]
S403
S404
ARE l0Cand AND l1Cand AVAILABLE AND (RefPicOrderCnt(refIdxL0l0Cand, L0) != RefPicOrderCnt(refIdxL1l1Cand, L1) OR mvL0l0Cand != mvL1l1Cand OR WpParam(refIdxL0l0Cand, L0) != WpParam(refIdxL1l1Cand, L1))?
NO
YES
S405
IS THERE NO COMBINATION OF l0Cand AND l1Cand, WHICH ARE THE SAME, WITHIN mergeCandList?
NO
YES
SET mergeCandList[numMergeCand] ← combCand, numMergeCand ← numMergeCand + 1, and combCnt < combCnt + 1
S406
SET combIdx ← combIdx + 1
S407
S408
combIdx == numInputMergeCand * (numInputMergeCand-1) OR numMergeCand == maxNumMergeCand OR combCnt == N?
NO
YES
END

FIG.35

combIdx	0	1	2	3	4	5	6	7	8	9	10	11
l0CandIdx	0	1	0	2	1	2	0	3	1	3	2	3
l1CandIdx	1	0	2	0	2	1	3	0	3	1	3	2

FIG.36

```
prediction_unit( x0, y0 ) {
    if( skip_flag[ x0 ][ y0 ] ) {
        merge_idx[ x0 ][ y0 ]
    } else if( PredMode  = =  MODE_INTRA ) {
        ...
    } else { /* MODE_INTER */
        if( entropy_coding_mode_flag || PartMode != PART_2Nx2N )
            merge_flag[ x0 ][ y0 ]
        if( merge_flag[ x0 ][ y0 ] ) {
            merge_idx[ x0 ][ y0 ]
        } else {
            ...
        }
    }
}
```

FIG.37

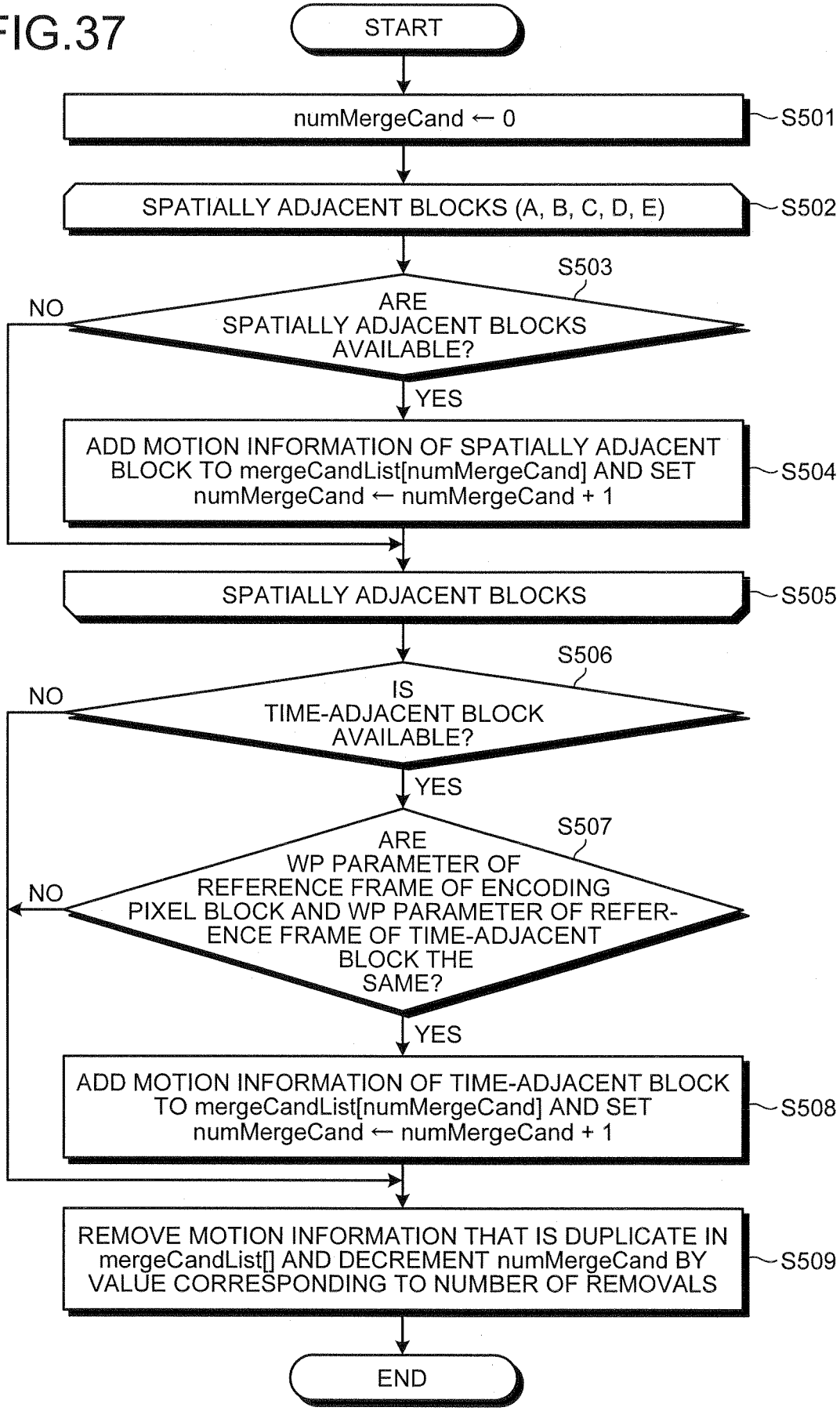

START
numMergeCand ← 0
S501
SPATIALLY ADJACENT BLOCKS (A, B, C, D, E)
S502
S503
ARE SPATIALLY ADJACENT BLOCKS AVAILABLE?
NO
YES
ADD MOTION INFORMATION OF SPATIALLY ADJACENT BLOCK TO mergeCandList[numMergeCand] AND SET numMergeCand ← numMergeCand + 1
S504
SPATIALLY ADJACENT BLOCKS
S505
S506
IS TIME-ADJACENT BLOCK AVAILABLE?
NO
YES
S507
ARE WP PARAMETER OF REFERENCE FRAME OF ENCODING PIXEL BLOCK AND WP PARAMETER OF REFERENCE FRAME OF TIME-ADJACENT BLOCK THE SAME?
NO
YES
ADD MOTION INFORMATION OF TIME-ADJACENT BLOCK TO mergeCandList[numMergeCand] AND SET numMergeCand ← numMergeCand + 1
S508
REMOVE MOTION INFORMATION THAT IS DUPLICATE IN mergeCandList[] AND DECREMENT numMergeCand BY VALUE CORRESPONDING TO NUMBER OF REMOVALS
S509
END

FIG.38

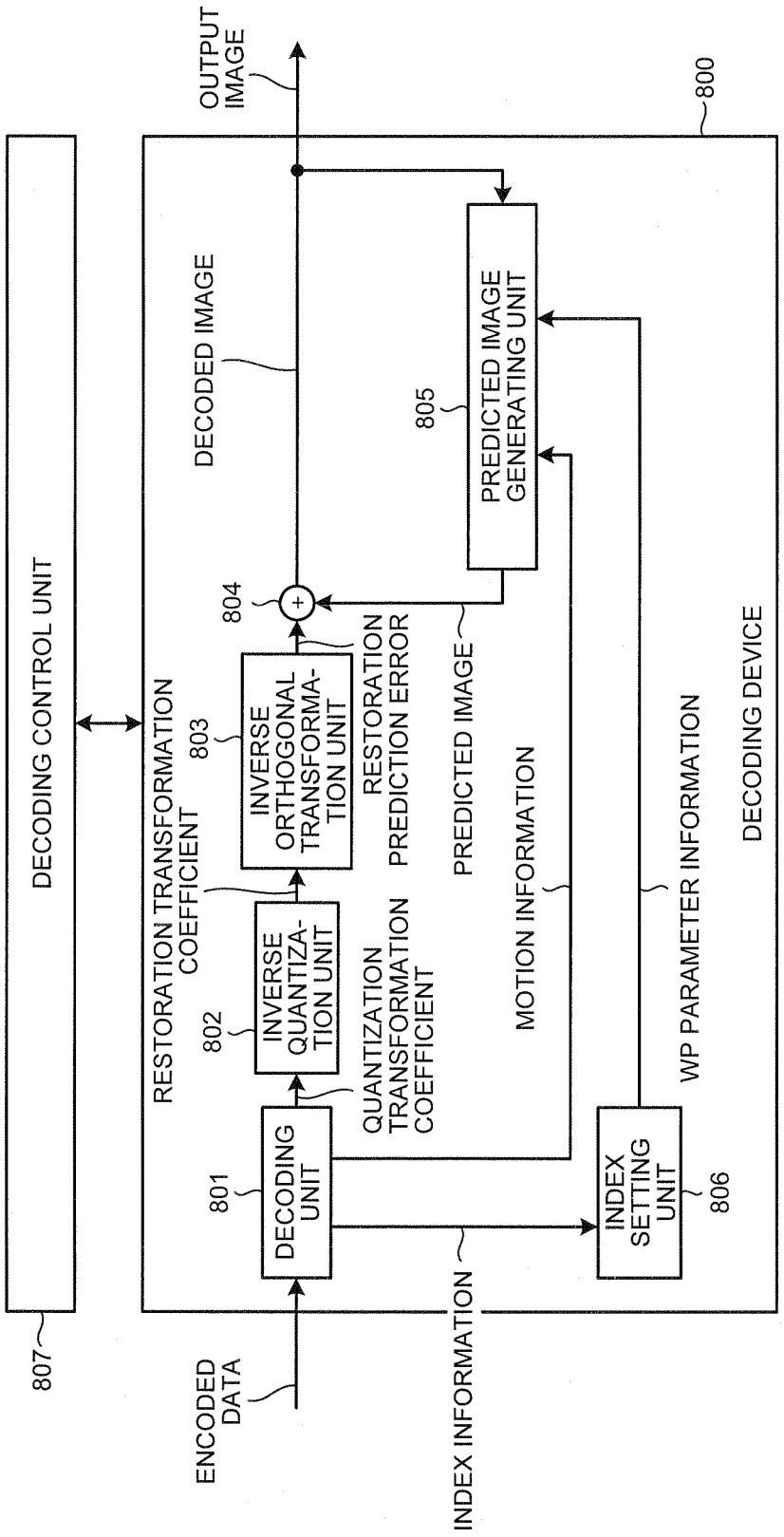

DECODING CONTROL UNIT
807
ENCODED DATA
DECODING UNIT
801
INVERSE QUANTIZA-TION UNIT
802
INVERSE ORTHOGONAL TRANSFORMA-TION UNIT
803
804
OUTPUT IMAGE
DECODED IMAGE
PREDICTED IMAGE GENERATING UNIT
805
INDEX SETTING UNIT
806
800
DECODING DEVICE
RESTORATION TRANSFORMATION COEFFICIENT
QUANTIZATION TRANSFORMATION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

FIG.39

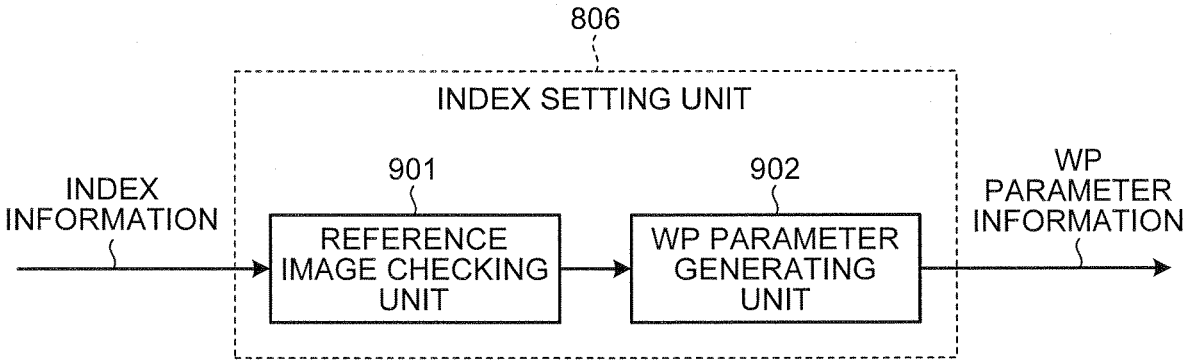

806
INDEX SETTING UNIT
901
902
INDEX INFORMATION
REFERENCE IMAGE CHECKING UNIT
WP PARAMETER GENERATING UNIT
WP PARAMETER INFORMATION

FIG.40

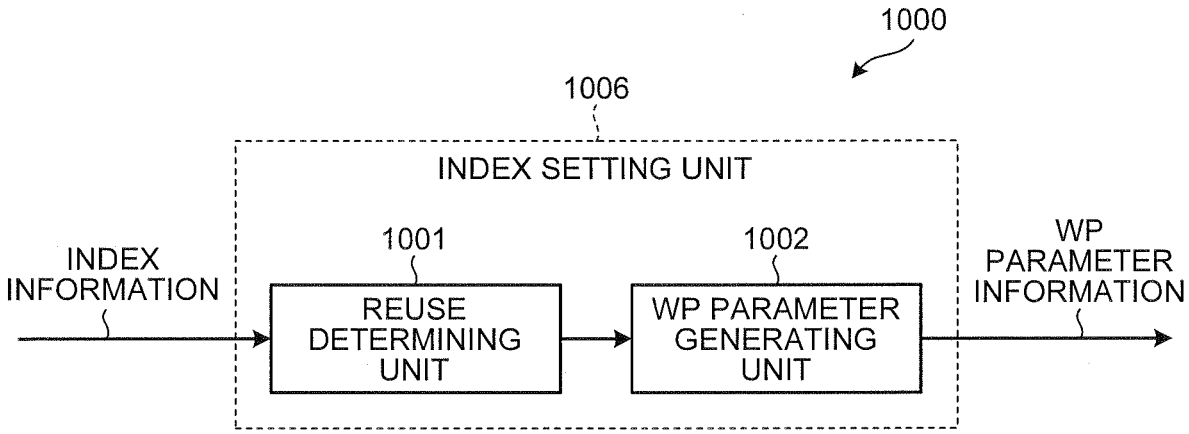
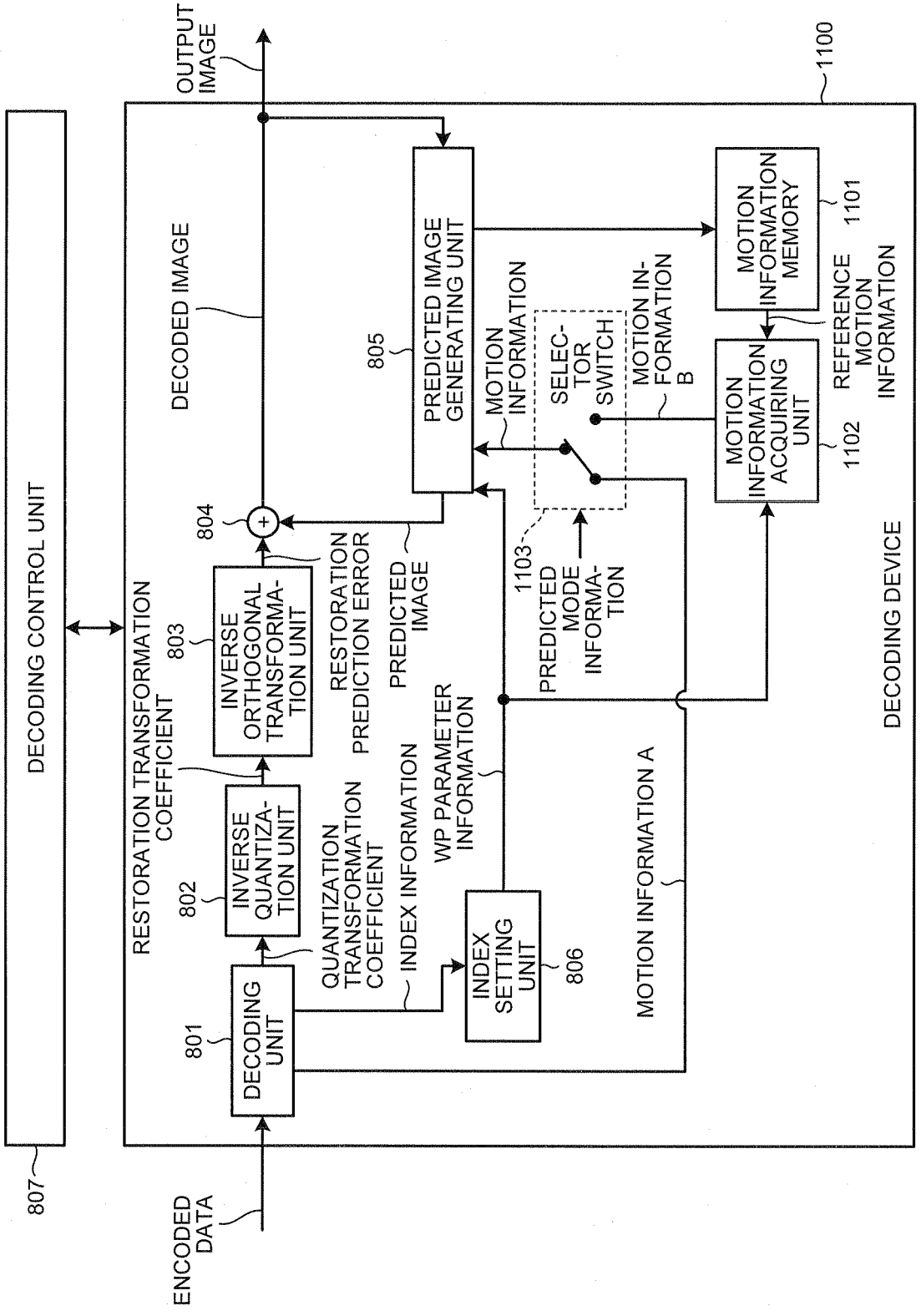

1000
1006
INDEX SETTING UNIT
1001
1002
INDEX INFORMATION
REUSE DETERMINING UNIT
WP PARAMETER GENERATING UNIT
WP PARAMETER INFORMATION

FIG.41


DECODING CONTROL UNIT
807
ENCODED DATA
DECODING UNIT
801
INVERSE QUANTIZATION UNIT
802
INVERSE ORTHOGONAL TRANSFORMATION UNIT
803
804
OUTPUT IMAGE
DECODED IMAGE
RESTORATION TRANSFORMATION COEFFICIENT
QUANTIZATION TRANSFORMATION COEFFICIENT
RESTORATION PREDICTION ERROR
INDEX INFORMATION
INDEX SETTING UNIT
806
WP PARAMETER INFORMATION
PREDICTED IMAGE
805
PREDICTED IMAGE GENERATING UNIT
MOTION INFORMATION
SELECTOR SWITCH
1103
PREDICTED MODE INFORMATION
MOTION INFORMATION A
MOTION INFORMATION B
MOTION INFORMATION ACQUIRING UNIT
1102
MOTION INFORMATION MEMORY
1101
REFERENCE MOTION INFORMATION
1100
DECODING DEVICE