

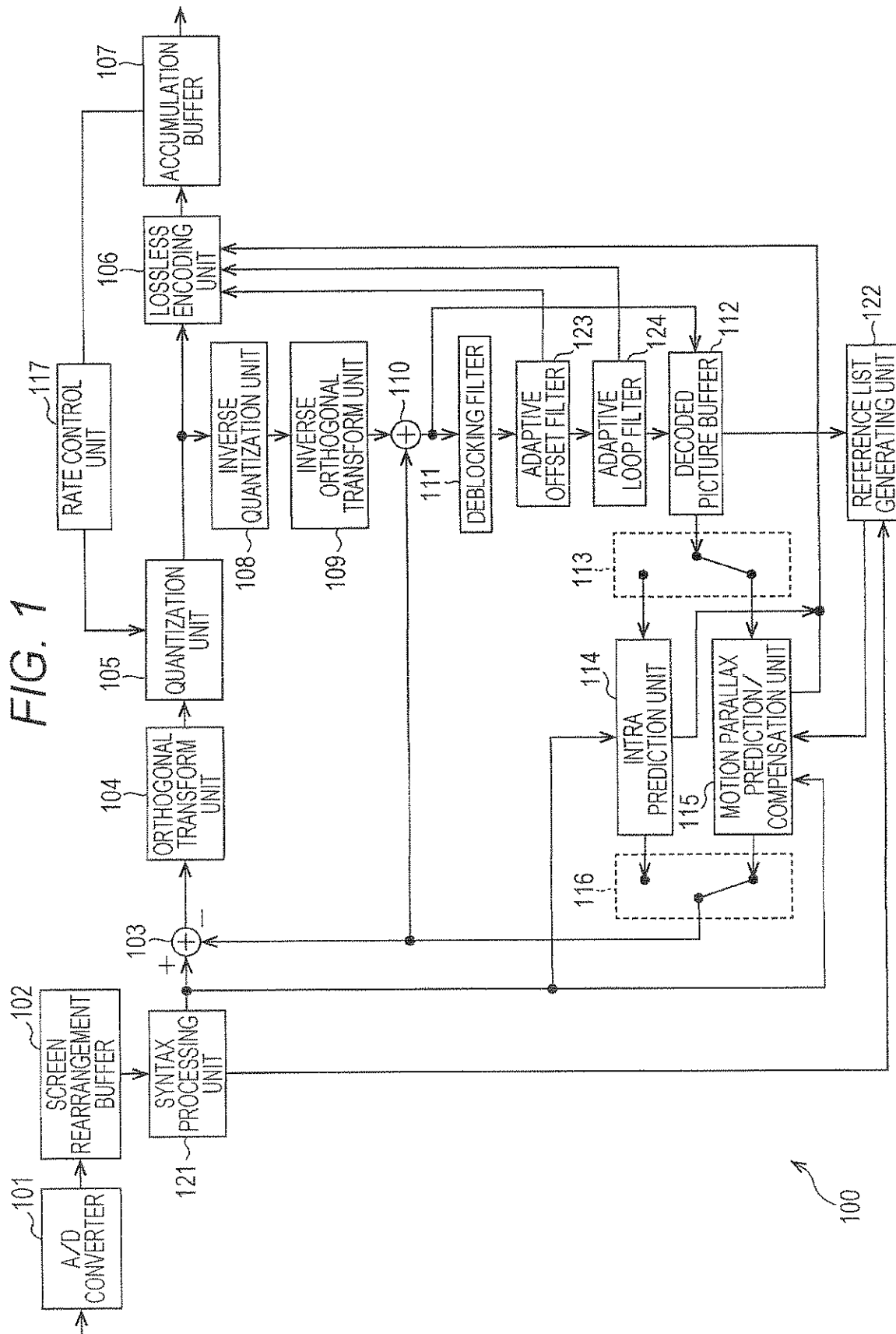


FIG. 2

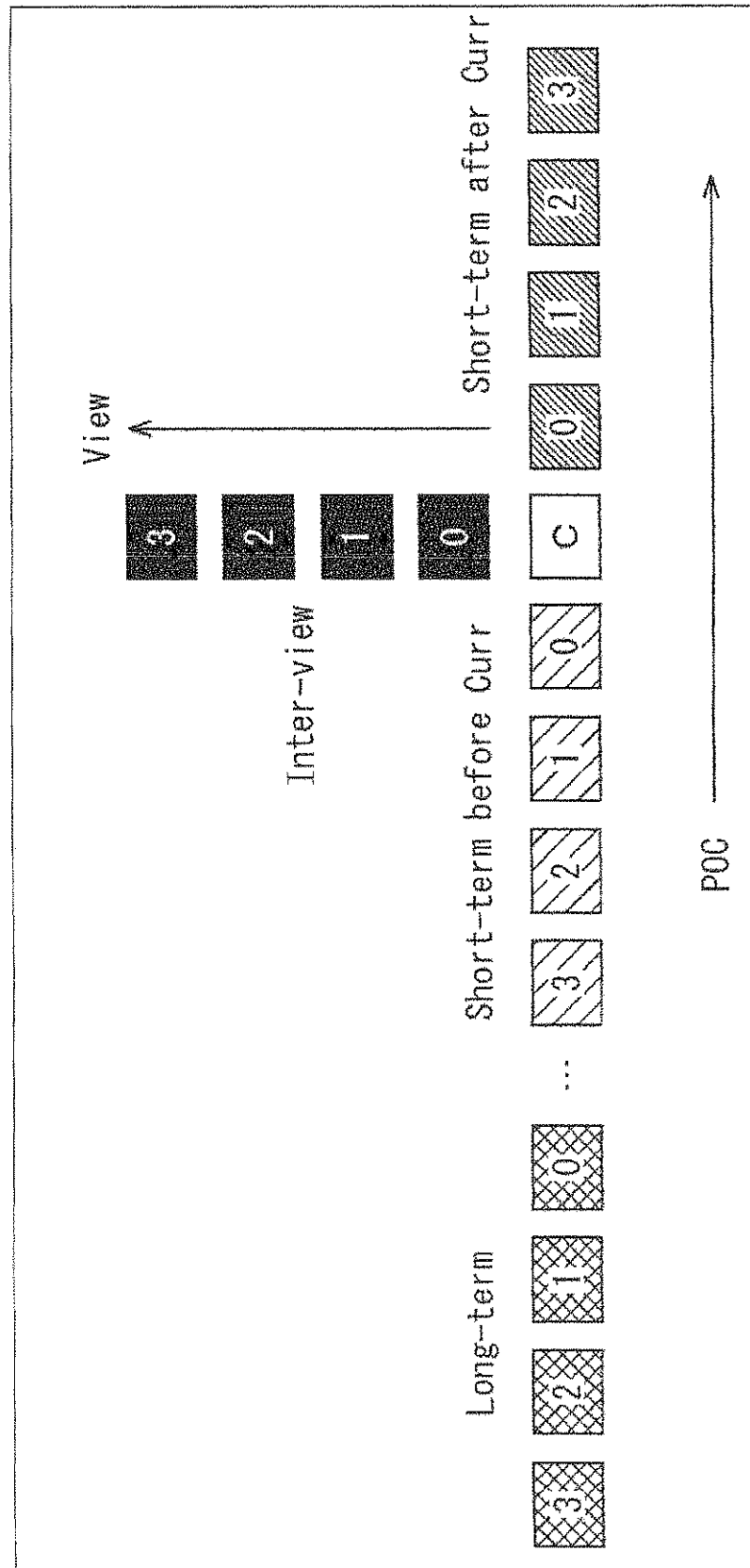


FIG. 3

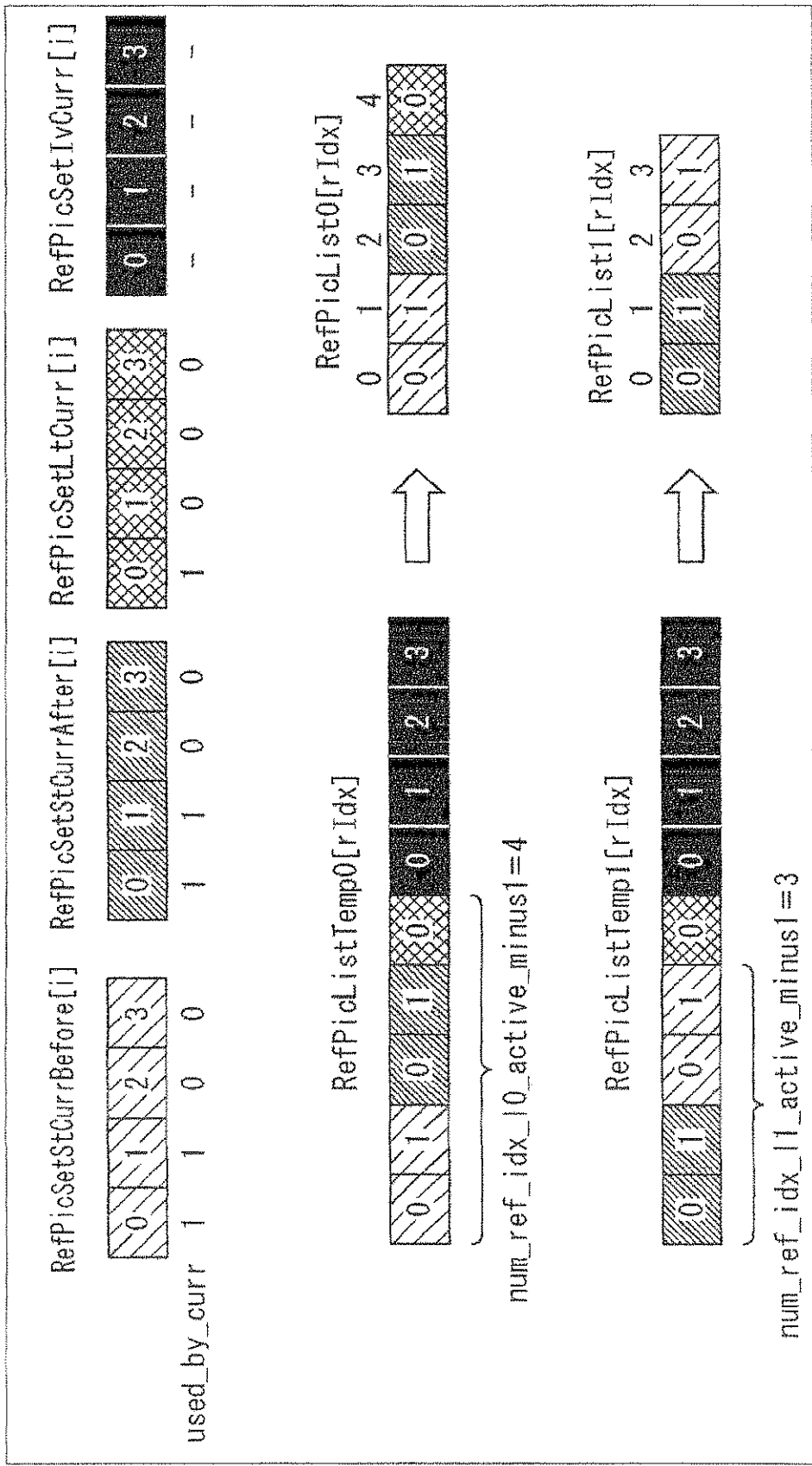
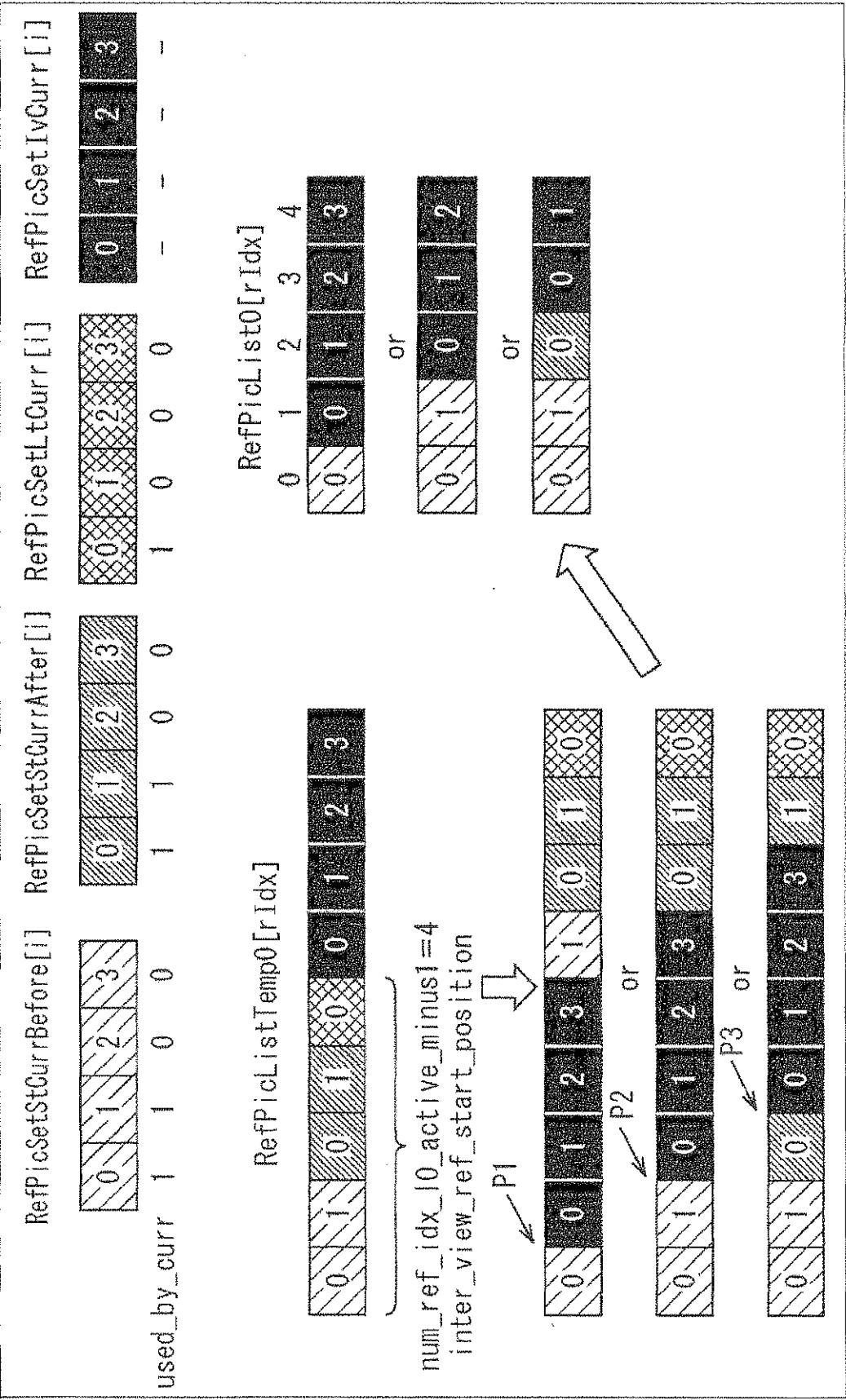


FIG. 4



5/28

FIG. 5

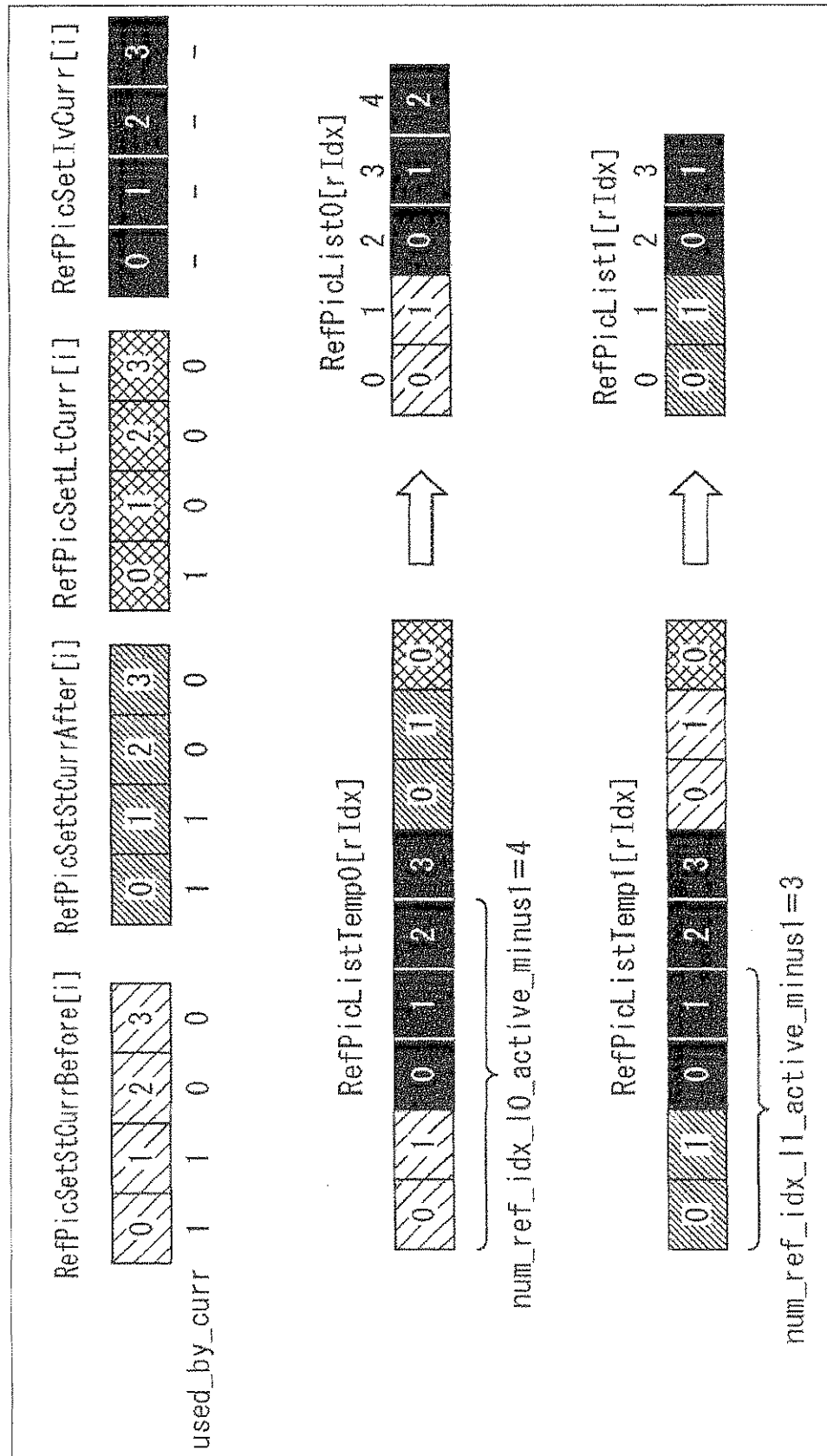
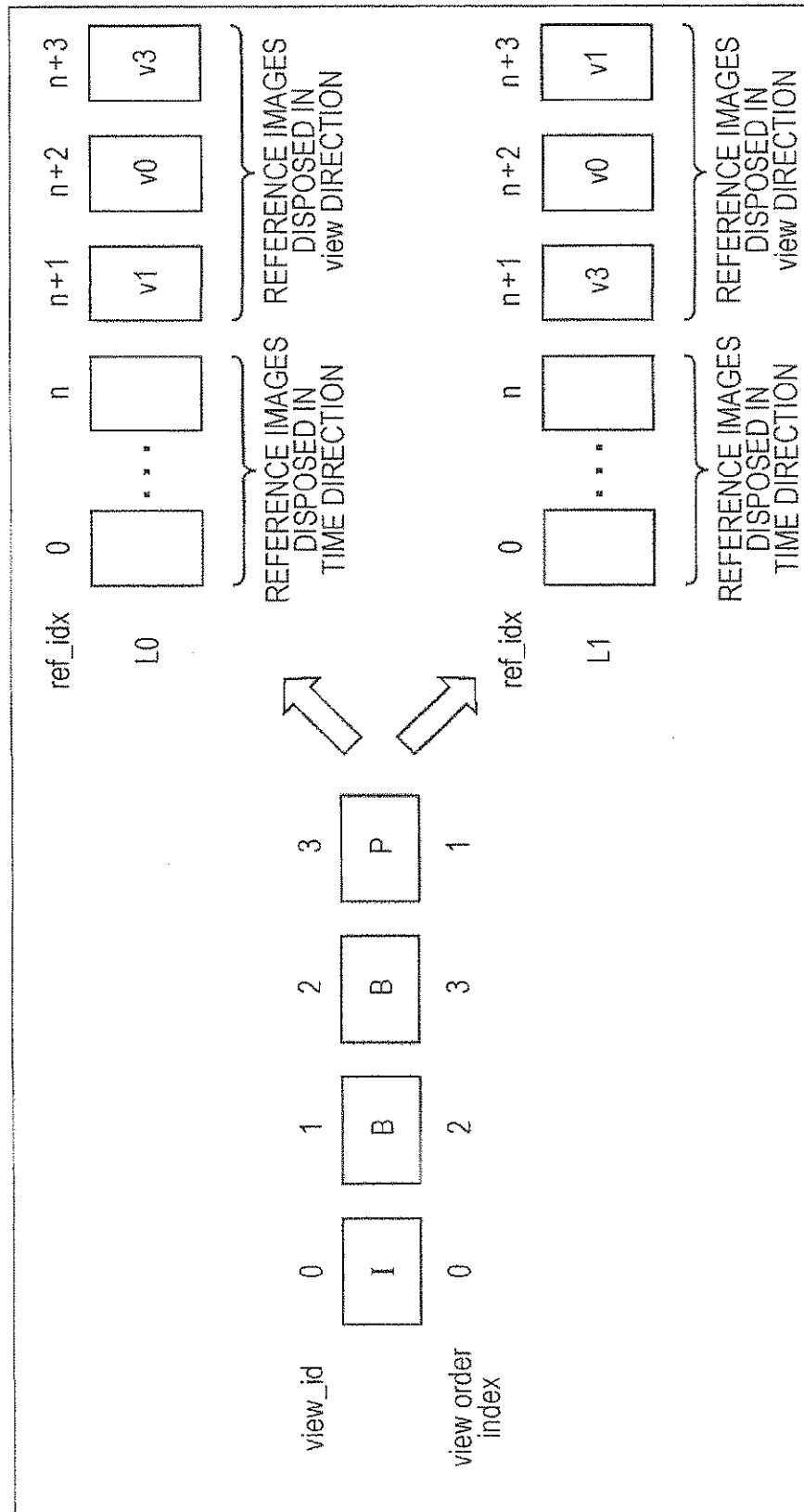


FIG. 7

		CONVENTIONAL TECHNOLOGY 1	CONVENTIONAL TECHNOLOGY 2	PRESENT TECHNOLOGY 1
PRESENT TECHNOLOGY 1	ADVANTAGE	<ul style="list-style-type: none"> • CODING EFFICIENCY CAN BE IMPROVED FOR SEQUENCE HAVING HIGH INTER-VIEW CORRELATION (GENERAL CASE). • WHEN NUMBER OF STbC AND STaC IS SMALL, NUMBER OF UNIQUE PICTURES CAN BE INCREASED IN WHOLE REFERENCE LIST L0/L1. 	<ul style="list-style-type: none"> • ADDITIONAL SYNTAX DOES NOT NEED TO BE TRANSMITTED. • PROCESSING AMOUNT IS SMALL OWING TO NO REARRANGEMENT OF TEMPORAL LIST. 	<div style="border: 1px solid black; height: 100px; width: 100%;"></div>
PRESENT TECHNOLOGY 2	ADVANTAGE	<ul style="list-style-type: none"> • WHEN TEMPORAL CORRELATION AND INTER-VIEW CORRELATION ARE ALTERNATELY ALIGNED, CODING EFFICIENCY CAN BE IMPROVED. • INTER-VIEW CAN BE ARRANGED AT SECOND POSITION FROM START OF REFERENCE LIST ALSO IN WORST CASE. 	<ul style="list-style-type: none"> • ADDITIONAL SYNTAX DOES NOT NEED TO BE TRANSMITTED. • WHEN TEMPORAL CORRELATION AND INTER-VIEW CORRELATION ARE ALTERNATELY ALIGNED, CODING EFFICIENCY CAN BE IMPROVED. 	<ul style="list-style-type: none"> • WHEN TEMPORAL CORRELATION AND INTER-VIEW CORRELATION ARE ALTERNATELY ALIGNED, CODING EFFICIENCY CAN BE IMPROVED. • INTER-VIEW CAN BE ARRANGED AT SECOND POSITION FROM START OF REFERENCE LIST ALSO IN WORST CASE.

FIG. 8



9/28

FIG. 9

```

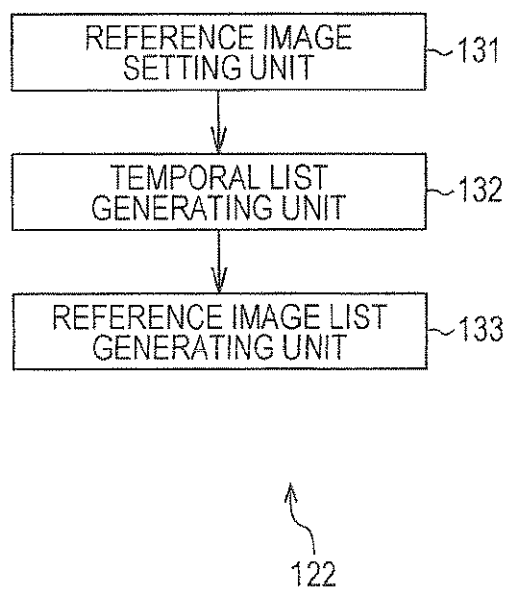
• The following applies:
• for ( i = 0; i < NumIvCurr; i++ )
  RefPicSetIvCurr[i] = "inter-view reference picture with layer identifier equal to LayerIdInterView[i]"

• rIdx = 0
  while ( rIdx < NumRpsCurrTempList0 ) {
    for ( i = 0; i < NumPocStCurrBefore && rIdx < NumRpsCurrTempList0; rIdx++, i++ )
      RefPicListTemp0[rIdx] = RefPicSetStCurrBefore[i]
    for ( i = 0; i < NumPocStCurrAfter && rIdx < NumRpsCurrTempList0; rIdx++, i++ )
      RefPicListTemp0[rIdx] = RefPicSetStCurrAfter[i]
    for ( i = 0; i < NumPocLtCurr && rIdx < NumRpsCurrTempList0; rIdx++, i++ )
      RefPicListTemp0[rIdx] = RefPicSetLtCurr[i]
    for ( i = 0; i < NumIvCurr && rIdx < NumRpsCurrTempList0; rIdx++, i++ )
      RefPicListTemp0[rIdx] = RefPicSetIvCurr[i]
  }

• rIdx = 0
  while ( rIdx < NumRpsCurrTempList1 ) {
    for ( i = 0; i < NumPocStCurrAfter && rIdx < NumRpsCurrTempList1; rIdx++, i++ )
      RefPicListTemp1[rIdx] = RefPicSetStCurrAfter[i]
    for ( i = 0; i < NumPocStCurrBefore && rIdx < NumRpsCurrTempList1; rIdx++, i++ )
      RefPicListTemp1[rIdx] = RefPicSetStCurrBefore[i]
    for ( i = 0; i < NumPocLtCurr && rIdx < NumRpsCurrTempList1; rIdx++, i++ )
      RefPicListTemp1[rIdx] = RefPicSetLtCurr[i]
    for ( i = 0; i < NumIvCurr && rIdx < NumRpsCurrTempList1; rIdx++, i++ )
      RefPicListTemp1[rIdx] = RefPicSetIvCurr[NumIvCurr-i]
  }

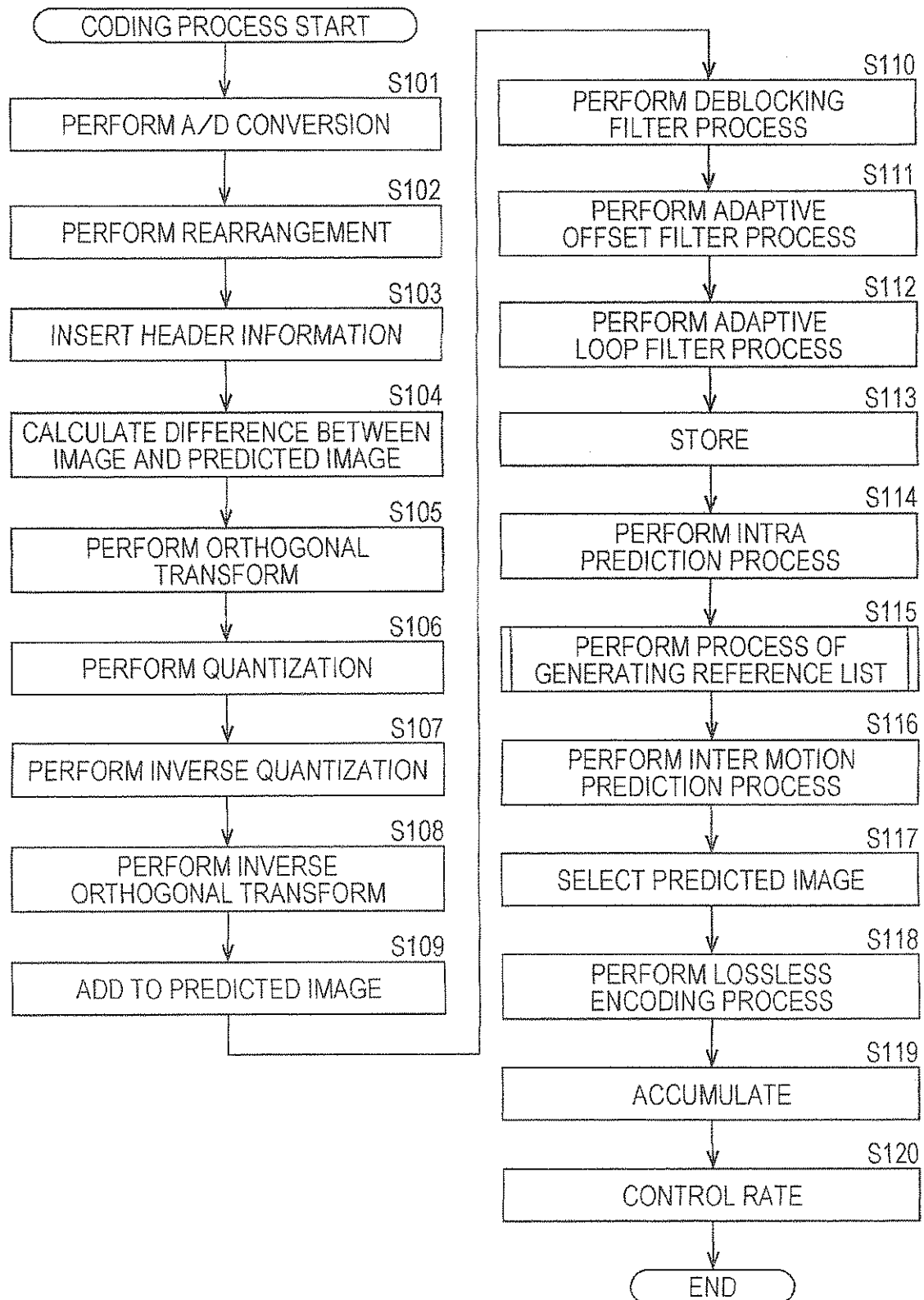
```

10/28

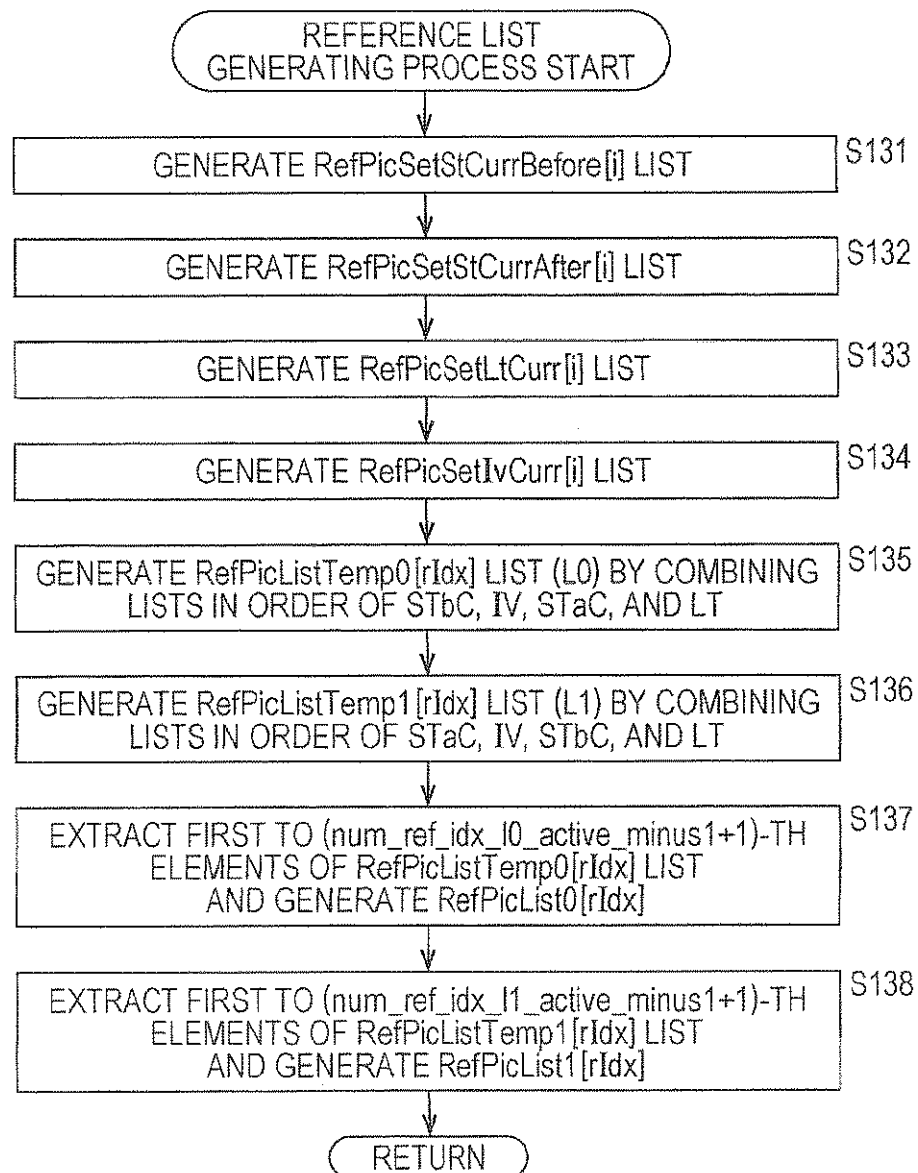
FIG. 10

11/28

FIG. 11



12/28

FIG. 12

13/28

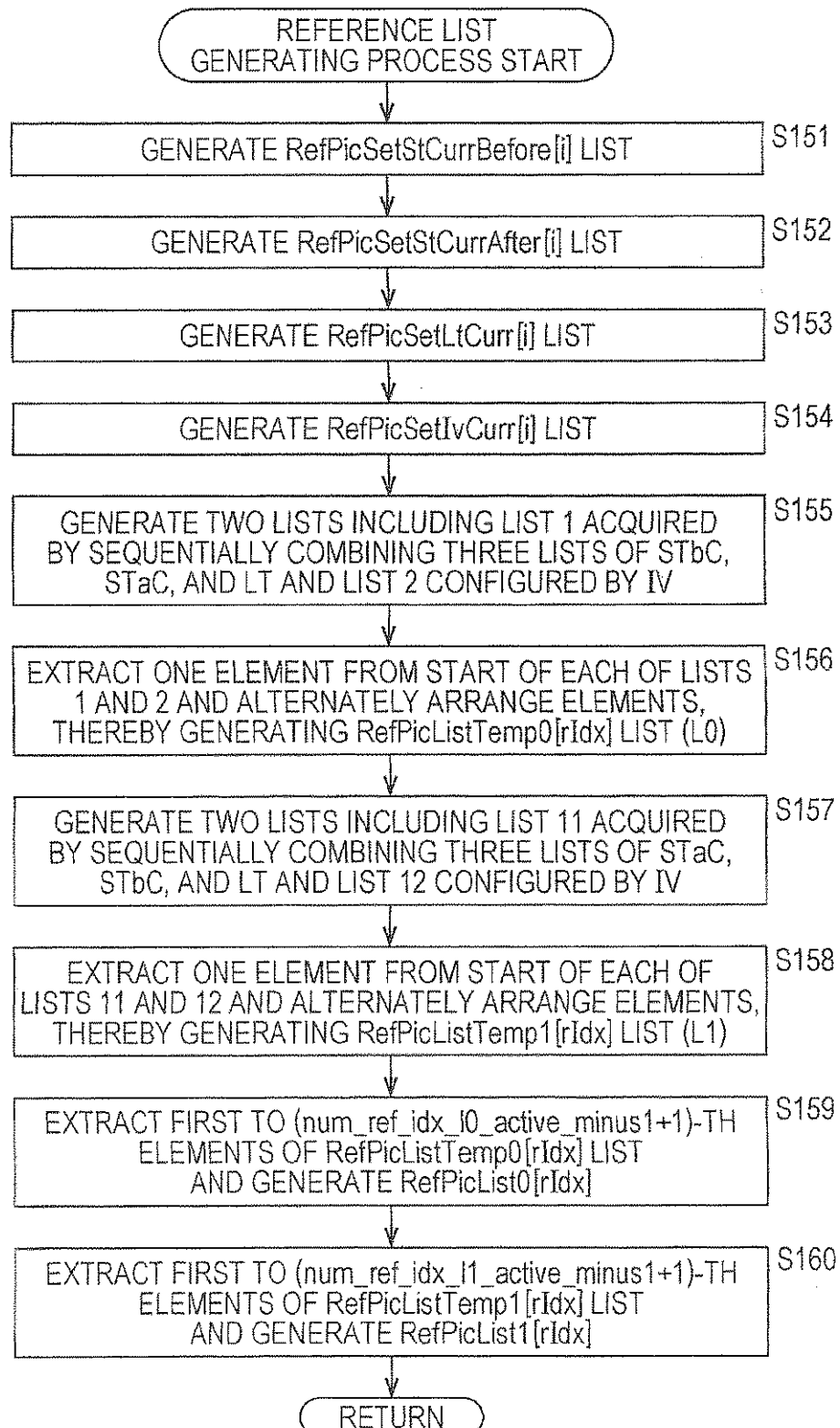
FIG. 13

FIG. 14

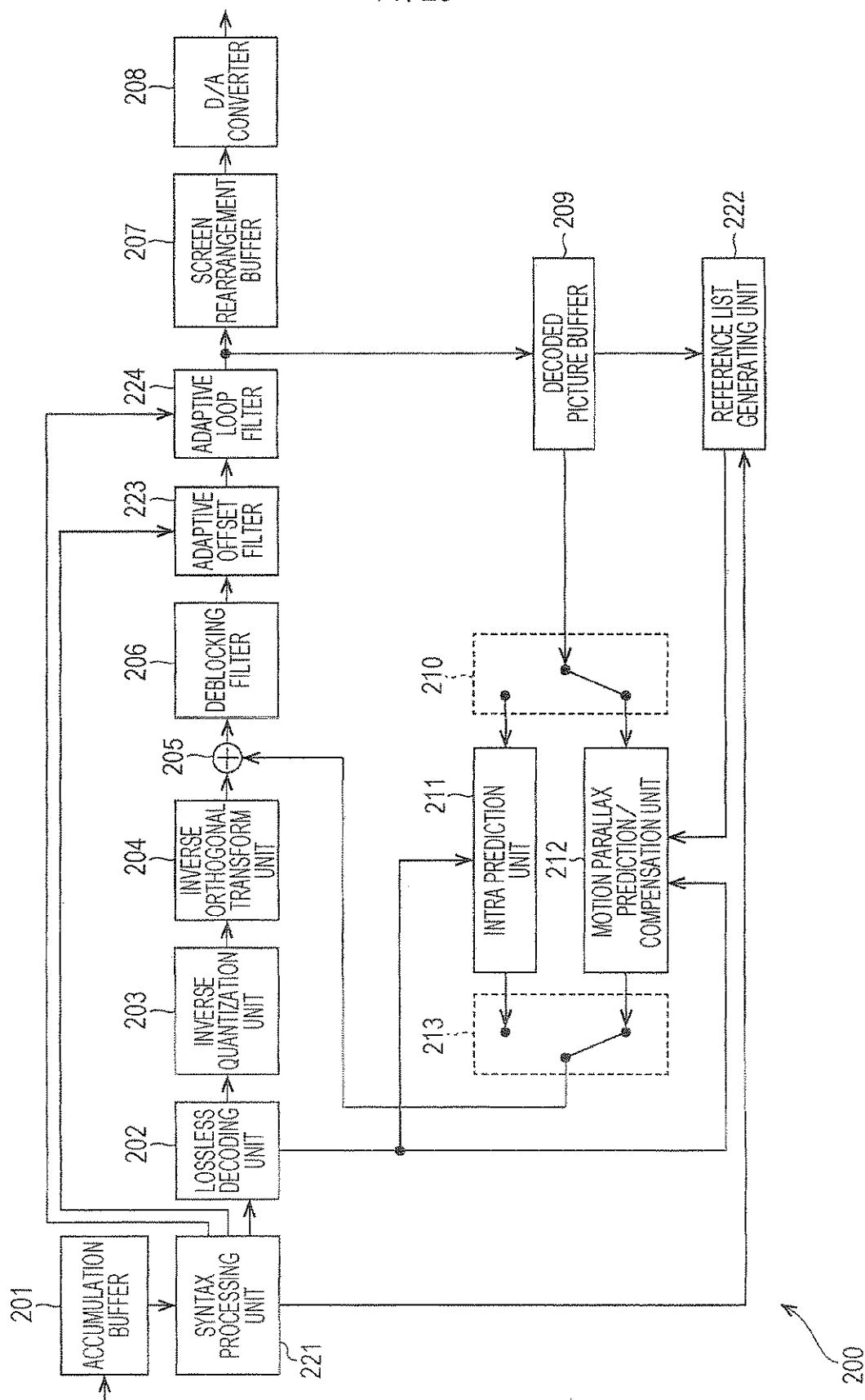
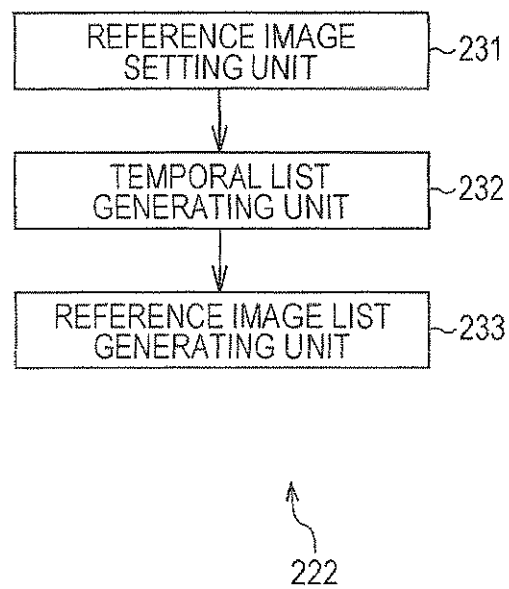
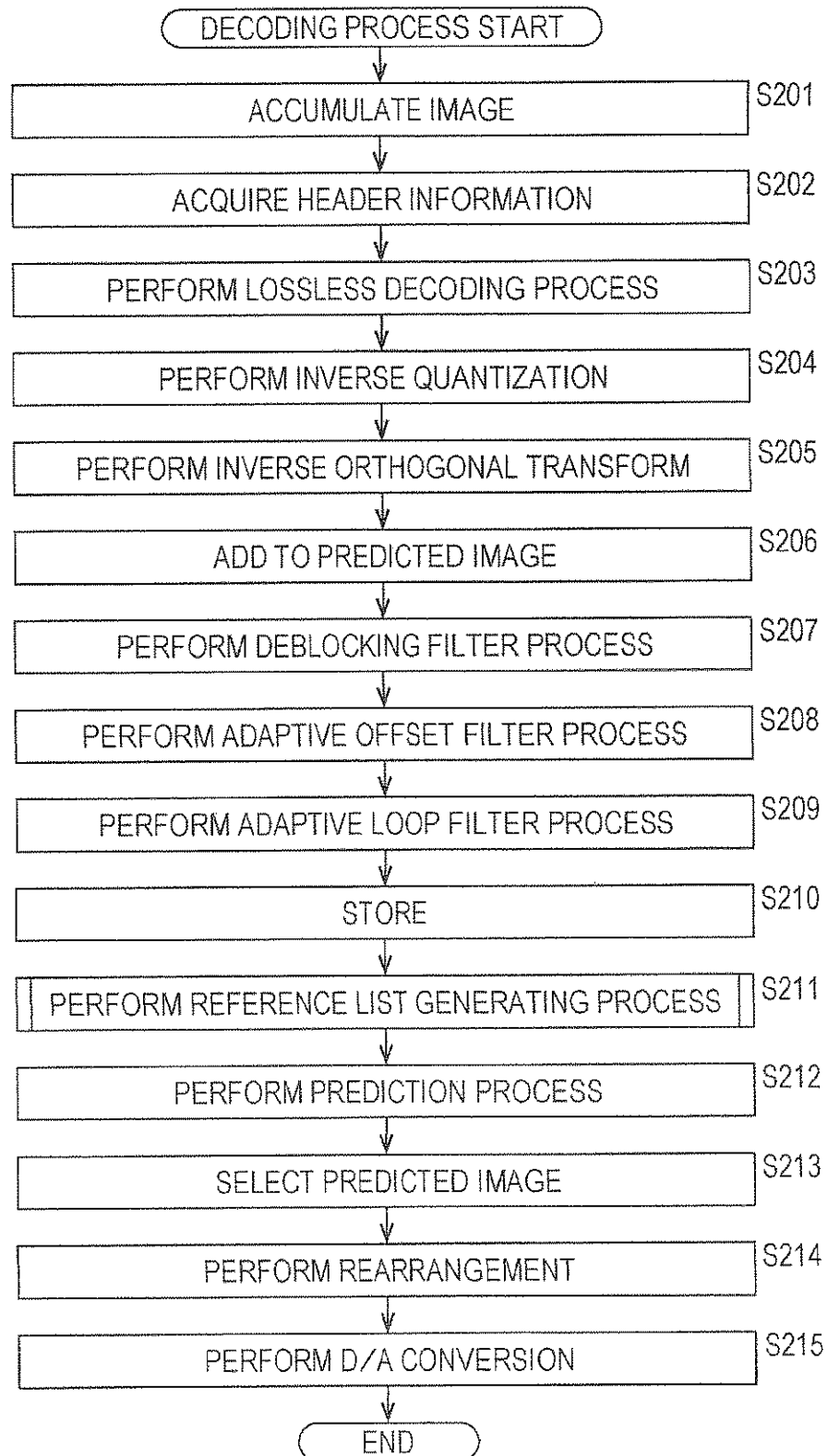


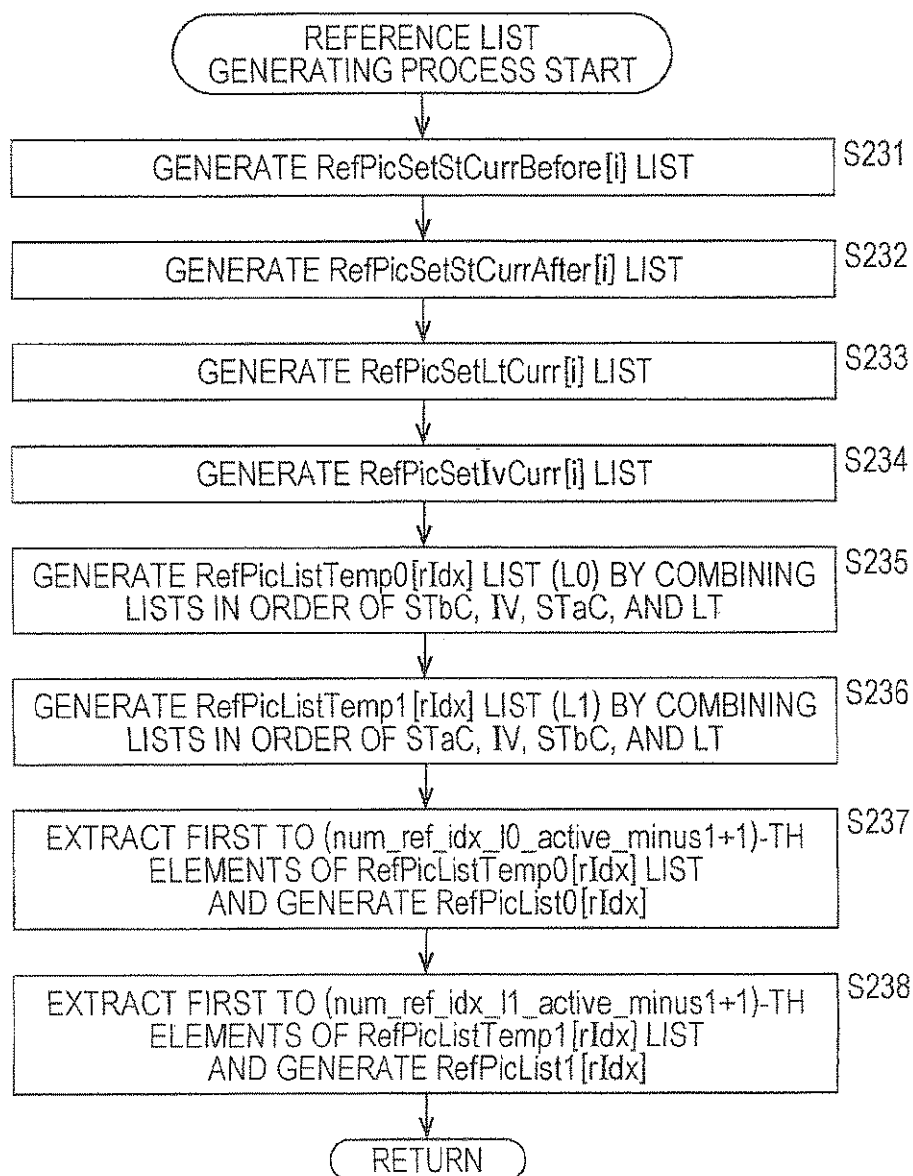
FIG. 15

16/28

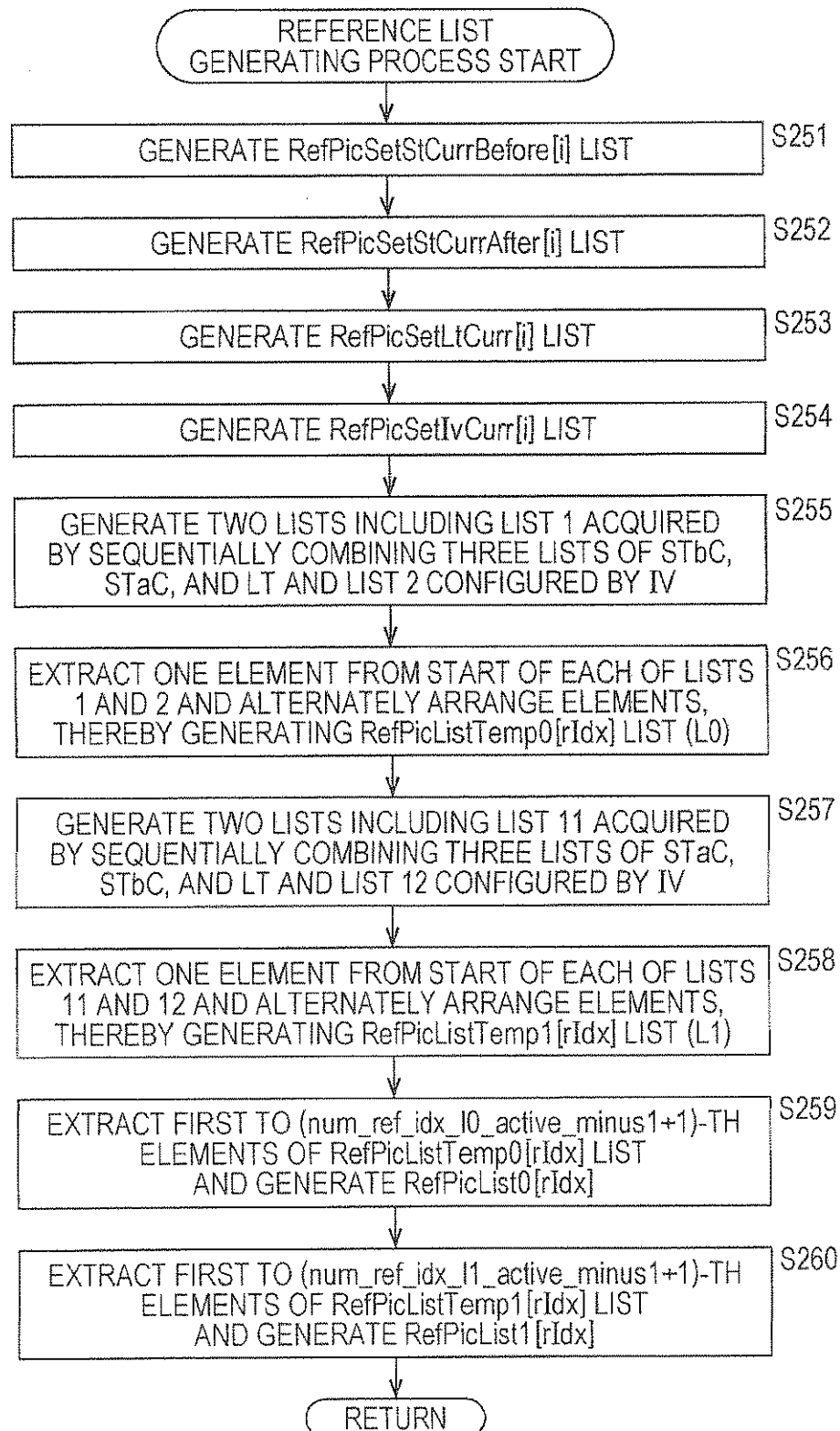
FIG. 16

17/28

FIG. 17



18/28

FIG. 18

19/28

FIG. 19

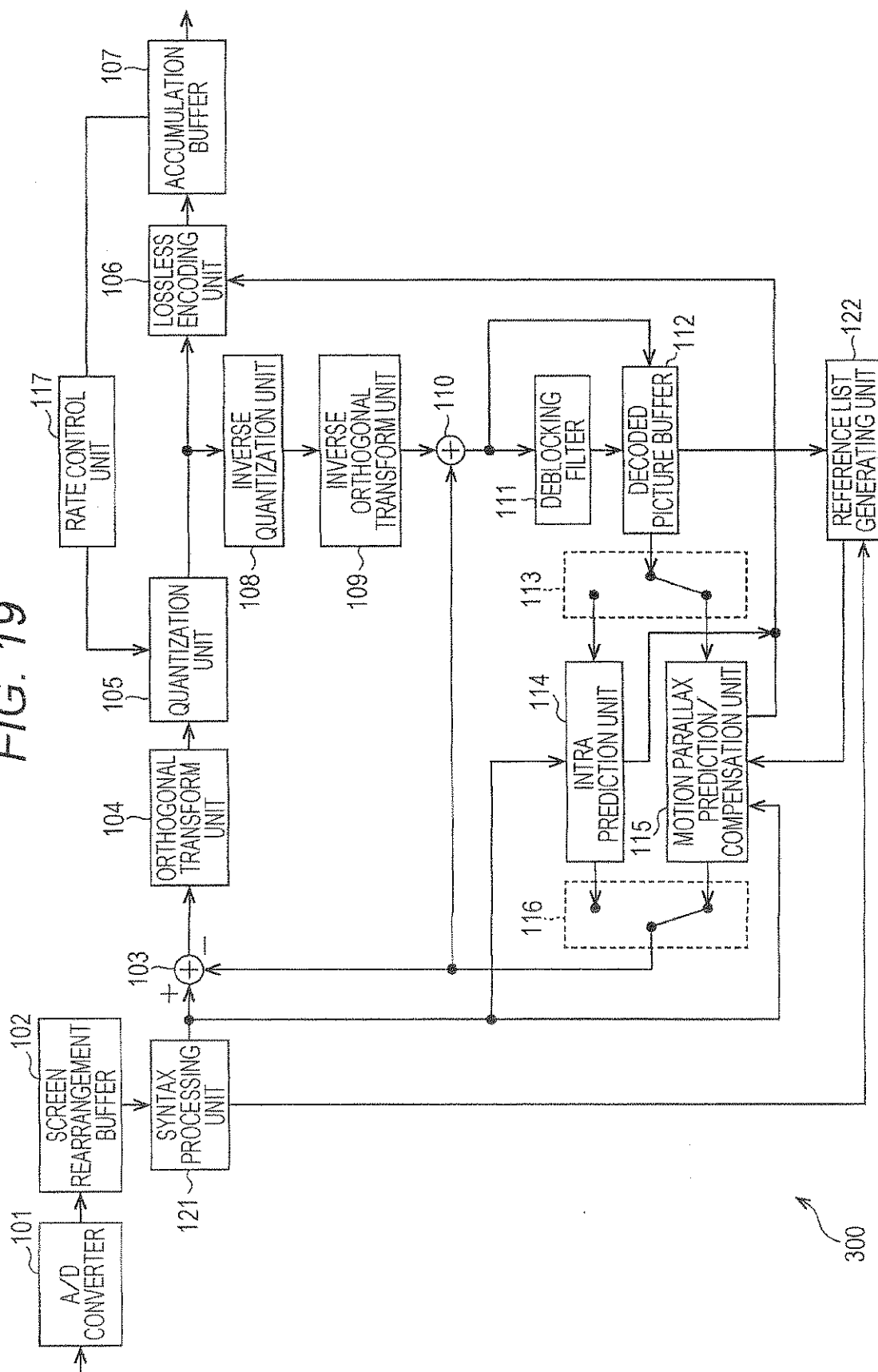


FIG. 20

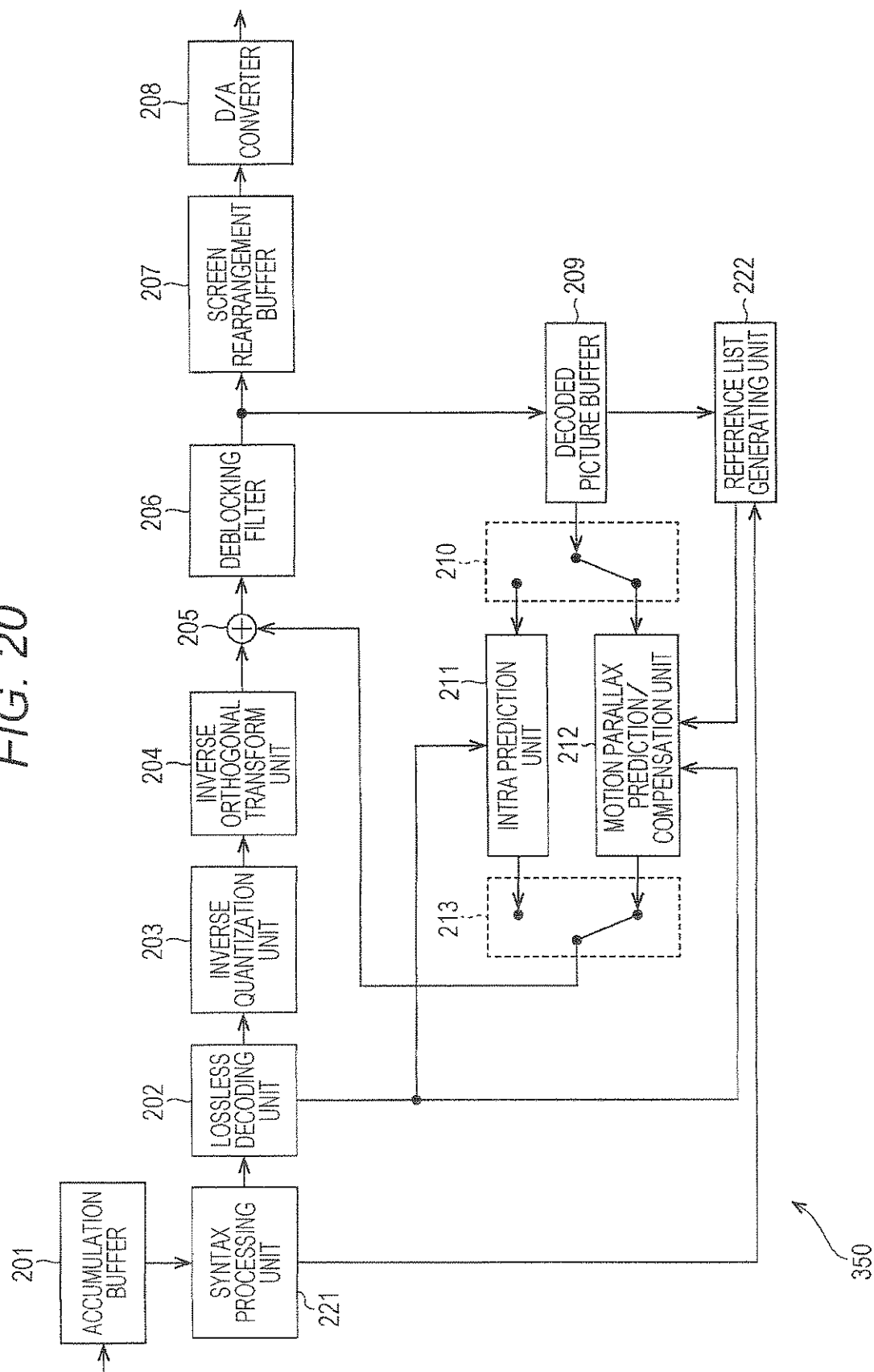


FIG. 21

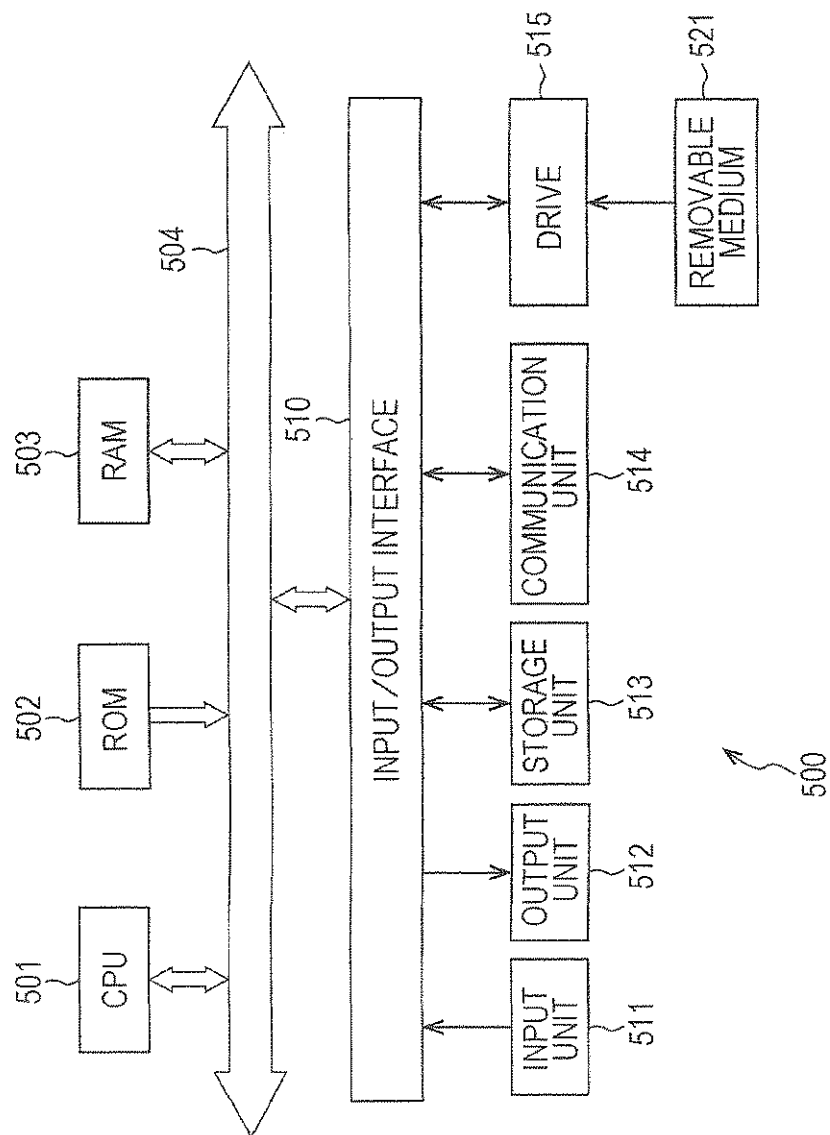


FIG. 22

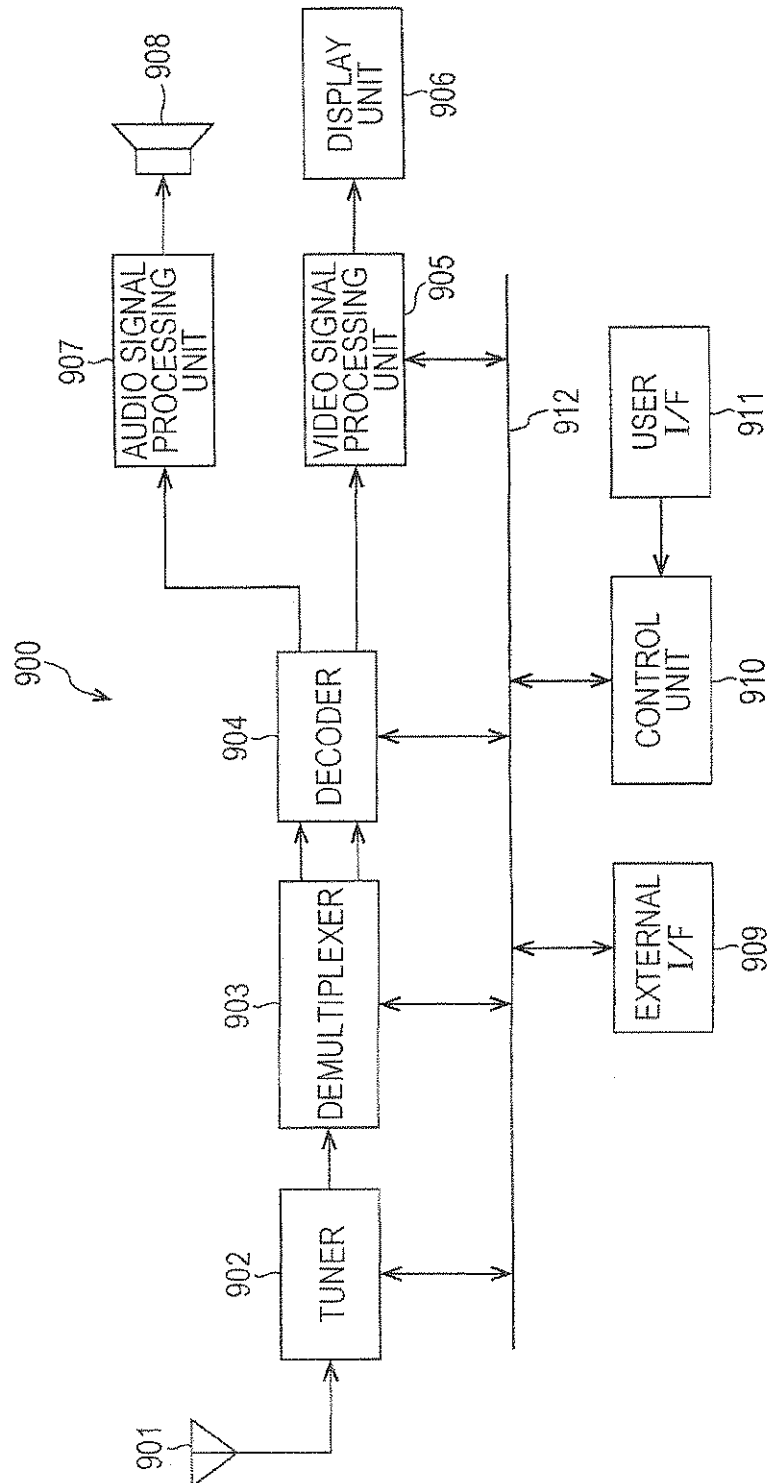
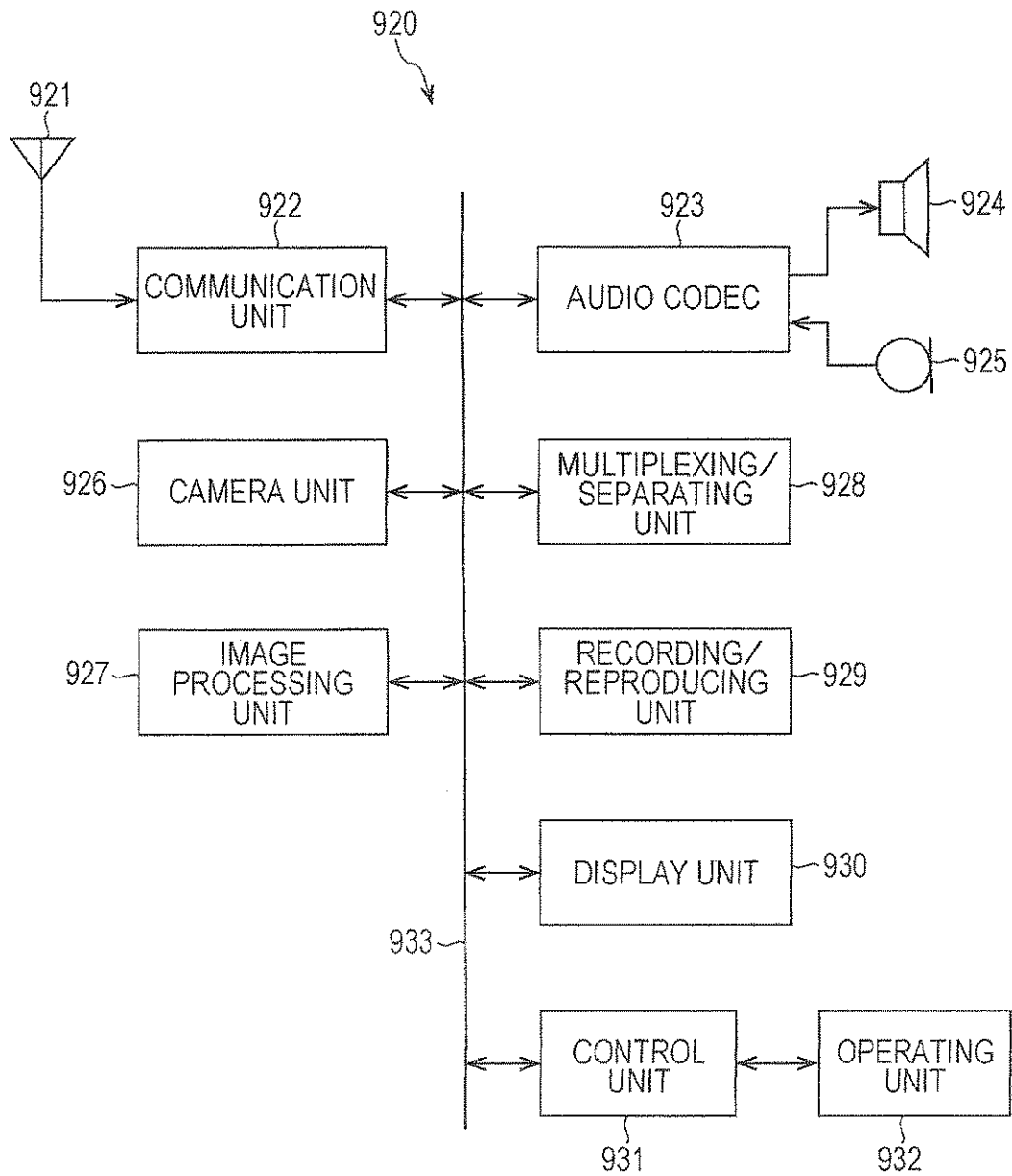


FIG. 23



24/28

FIG. 24

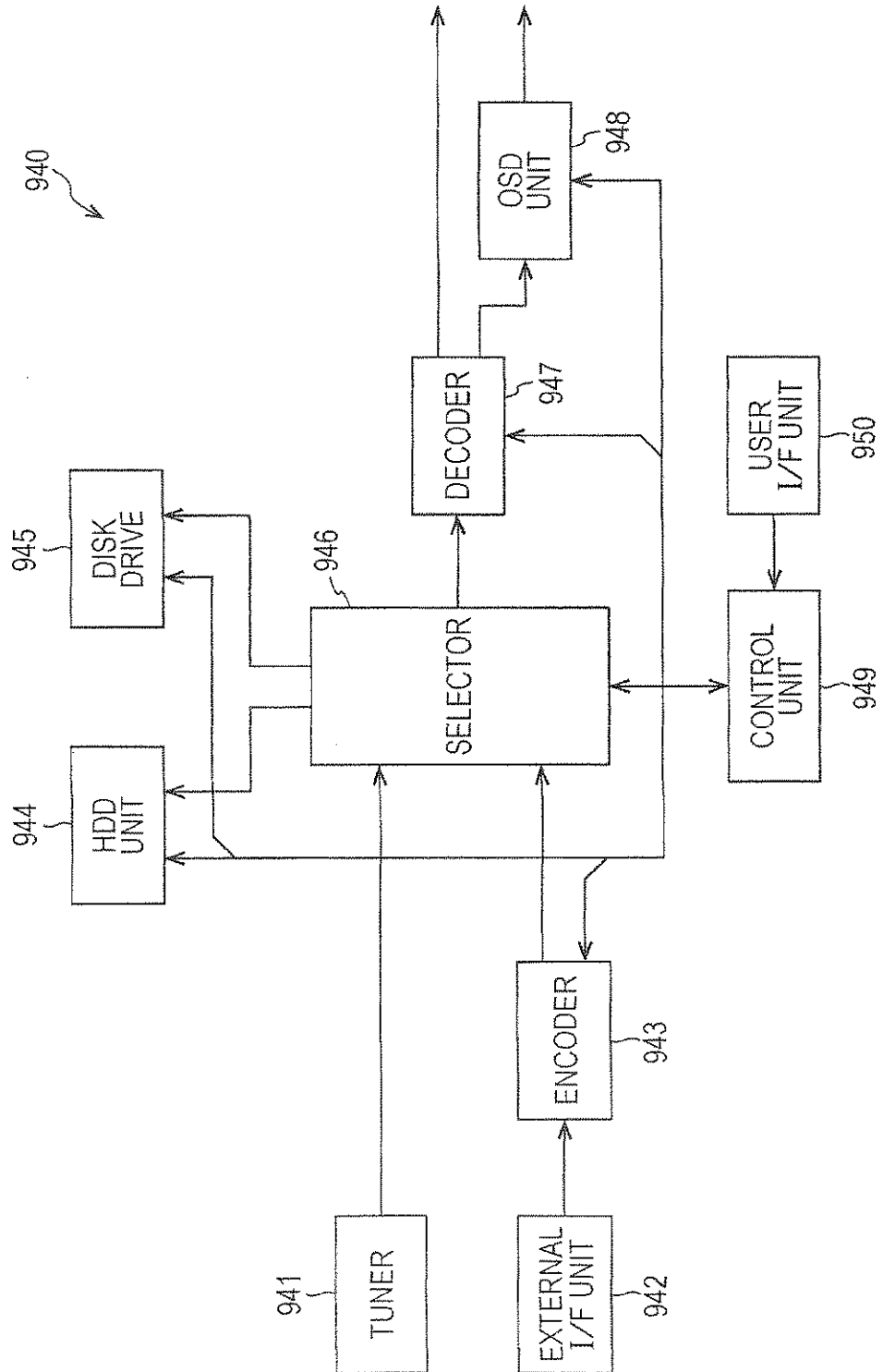


FIG. 25

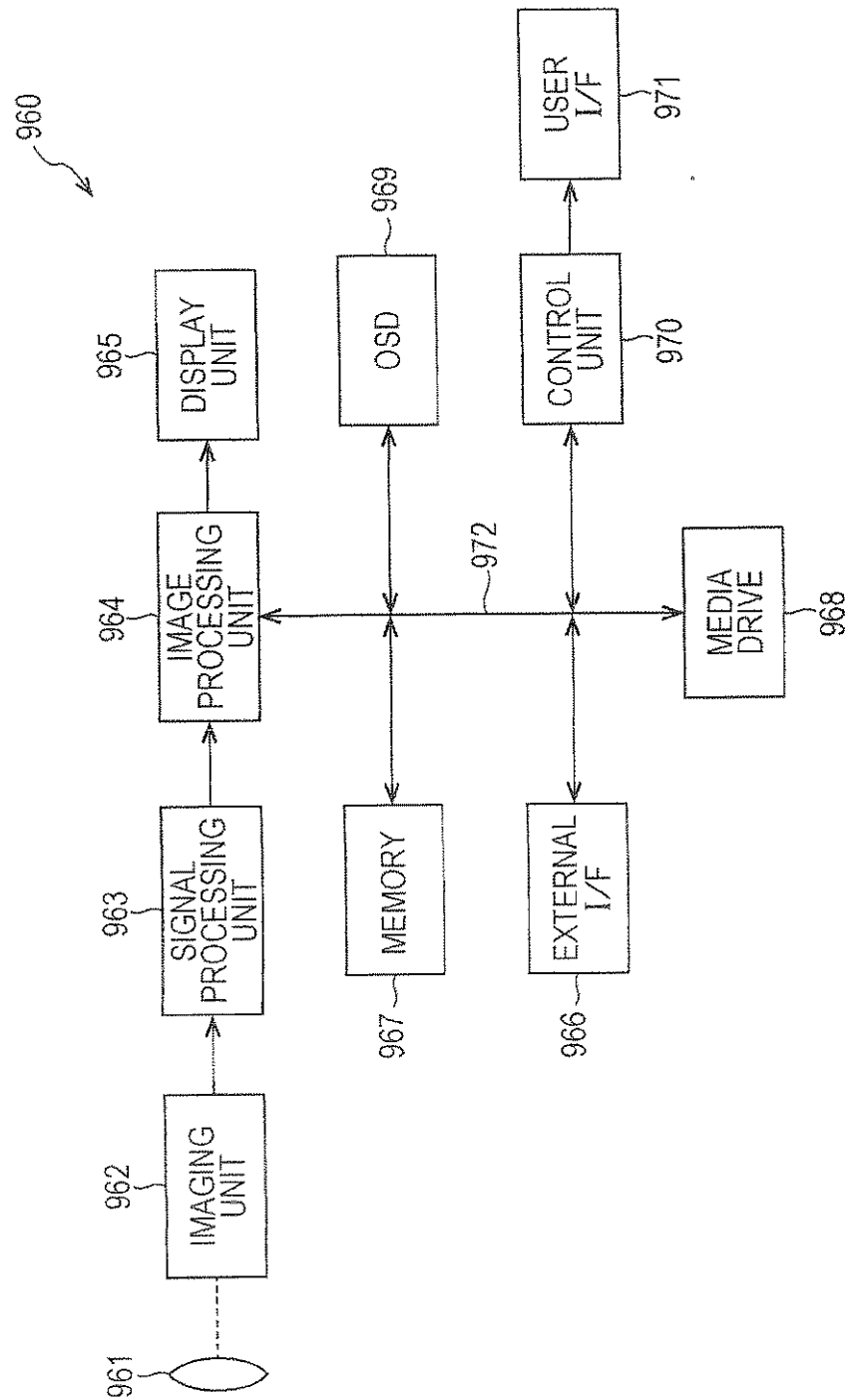


FIG. 26

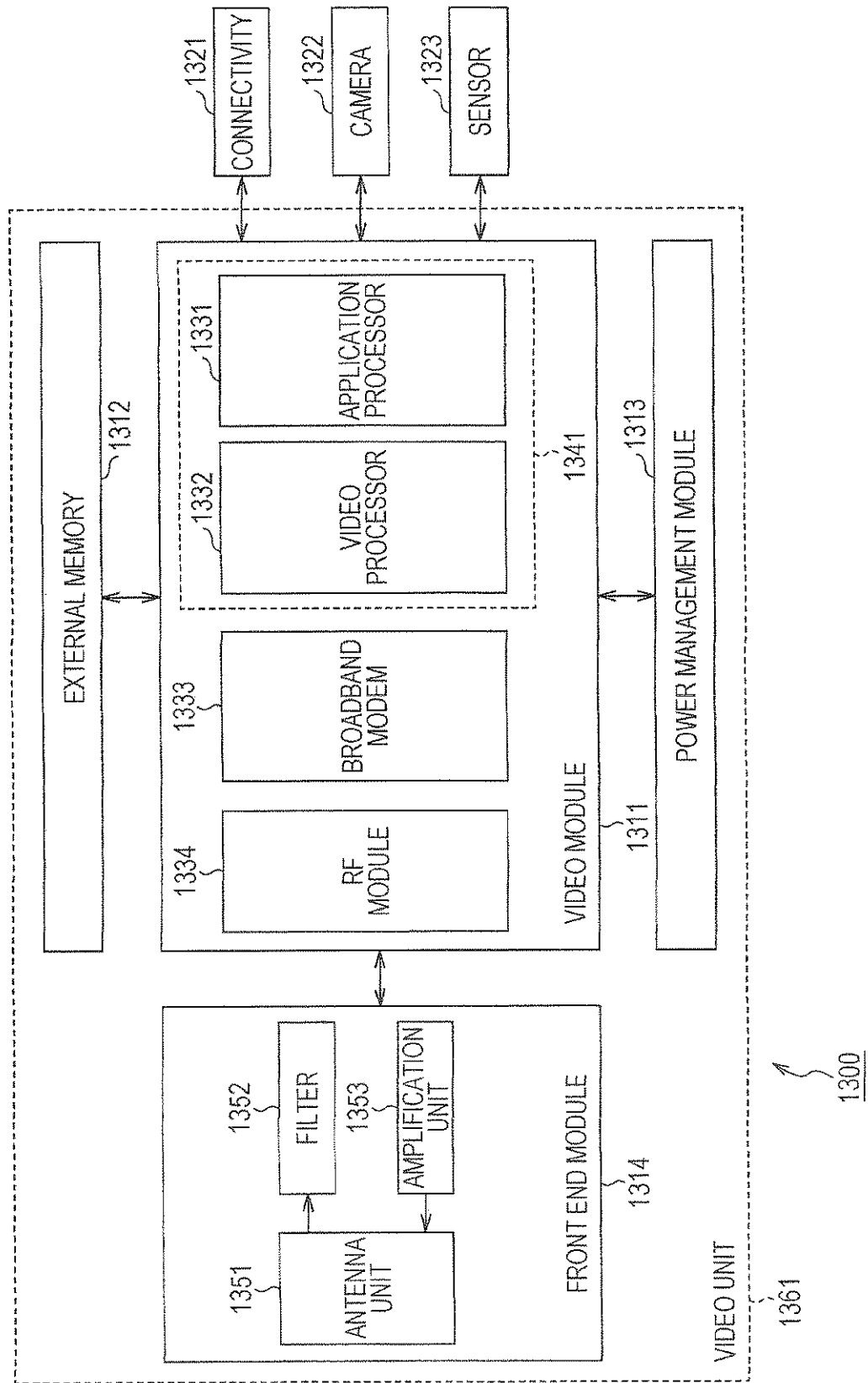


FIG. 27

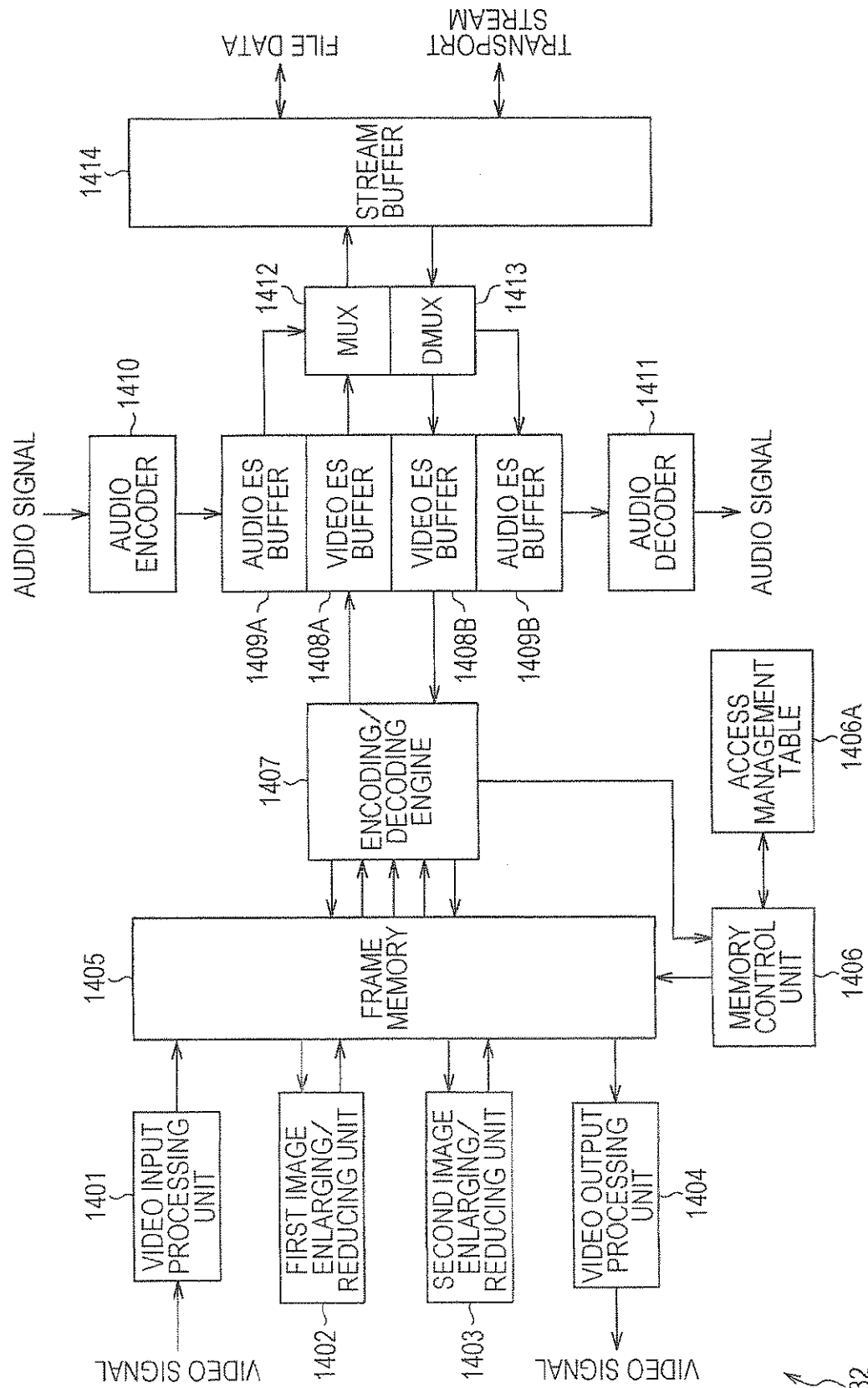


FIG. 28

