

1/23

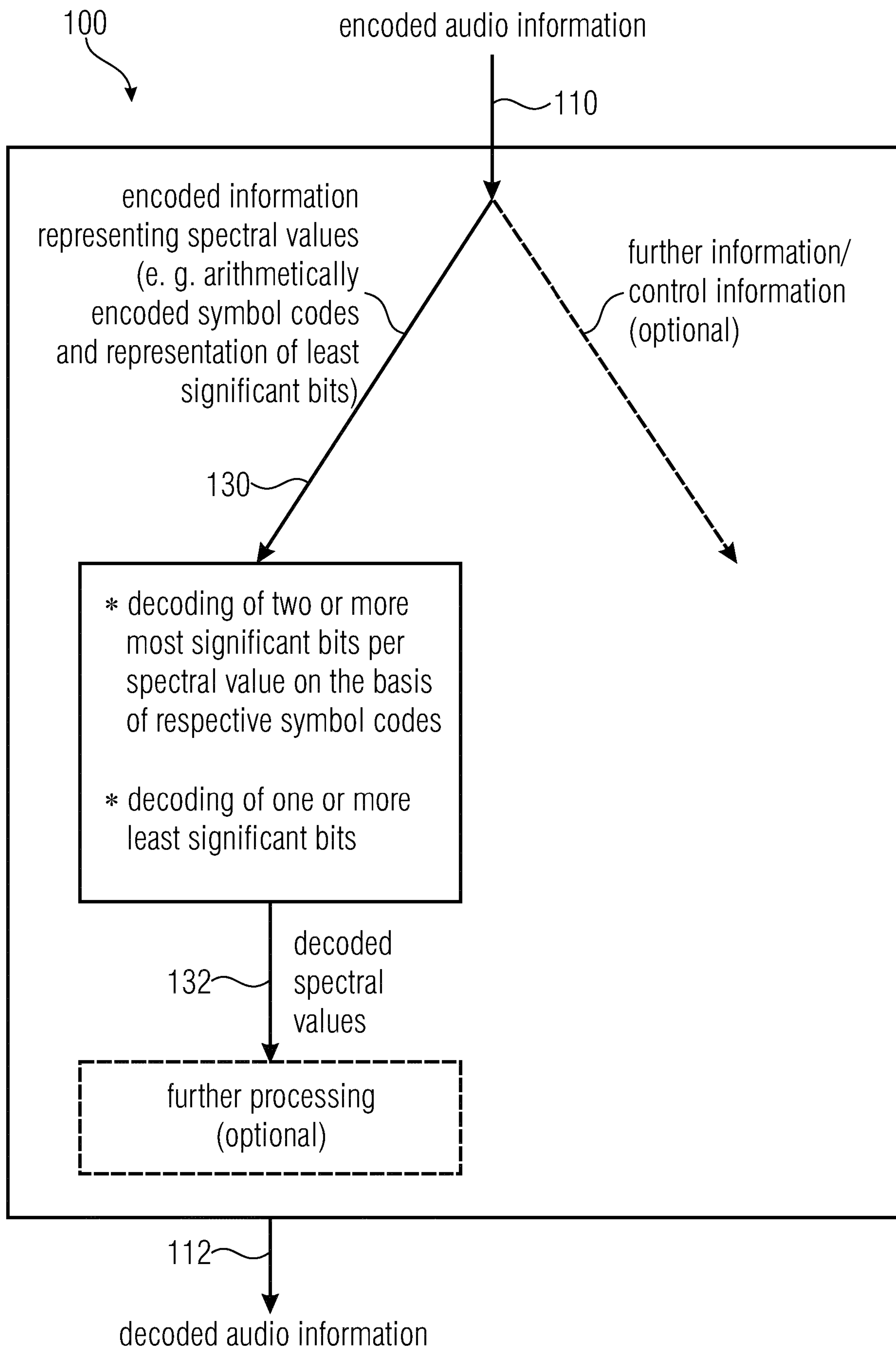


Fig. 1

2/23

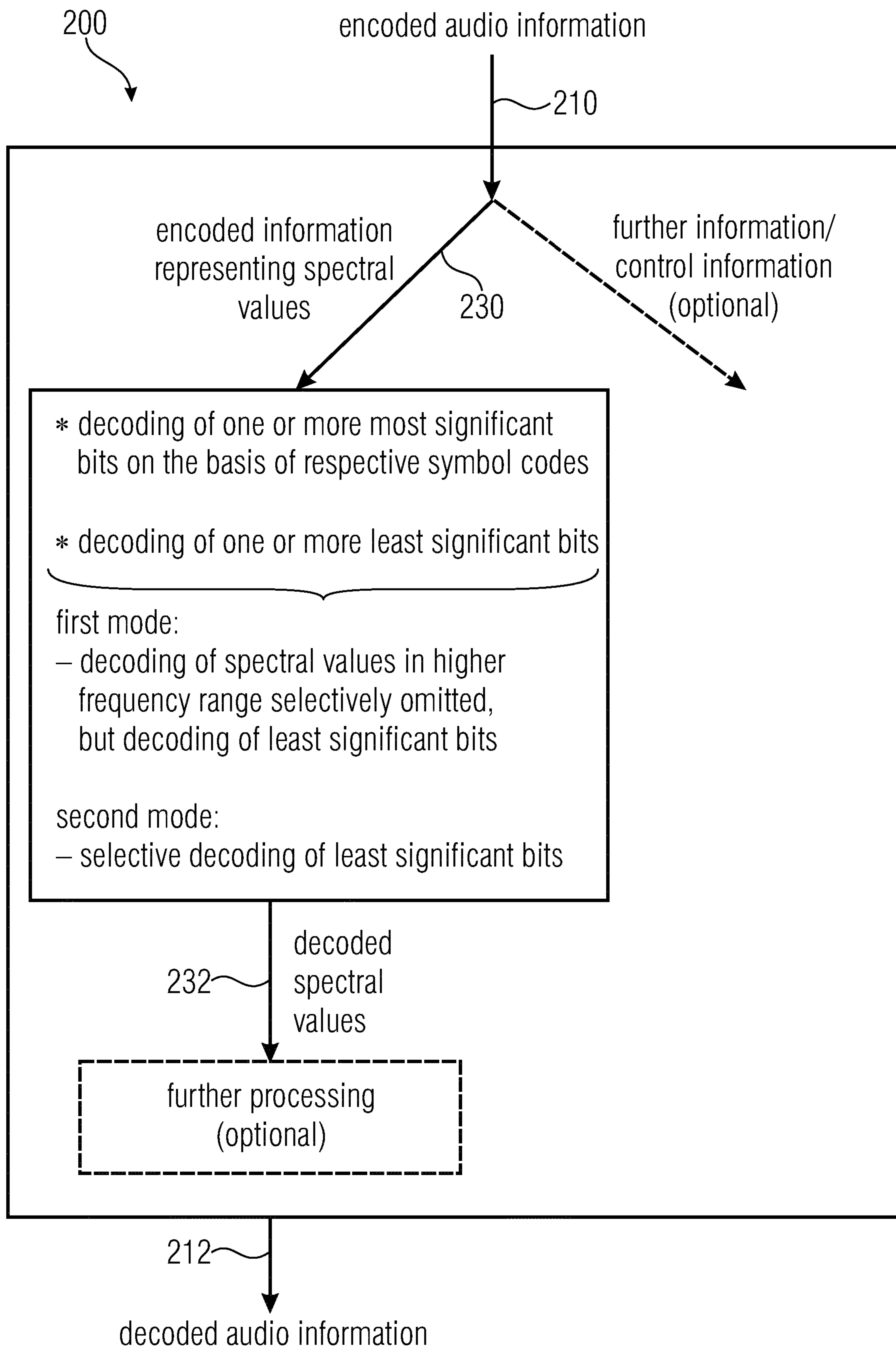


Fig. 2

3/23

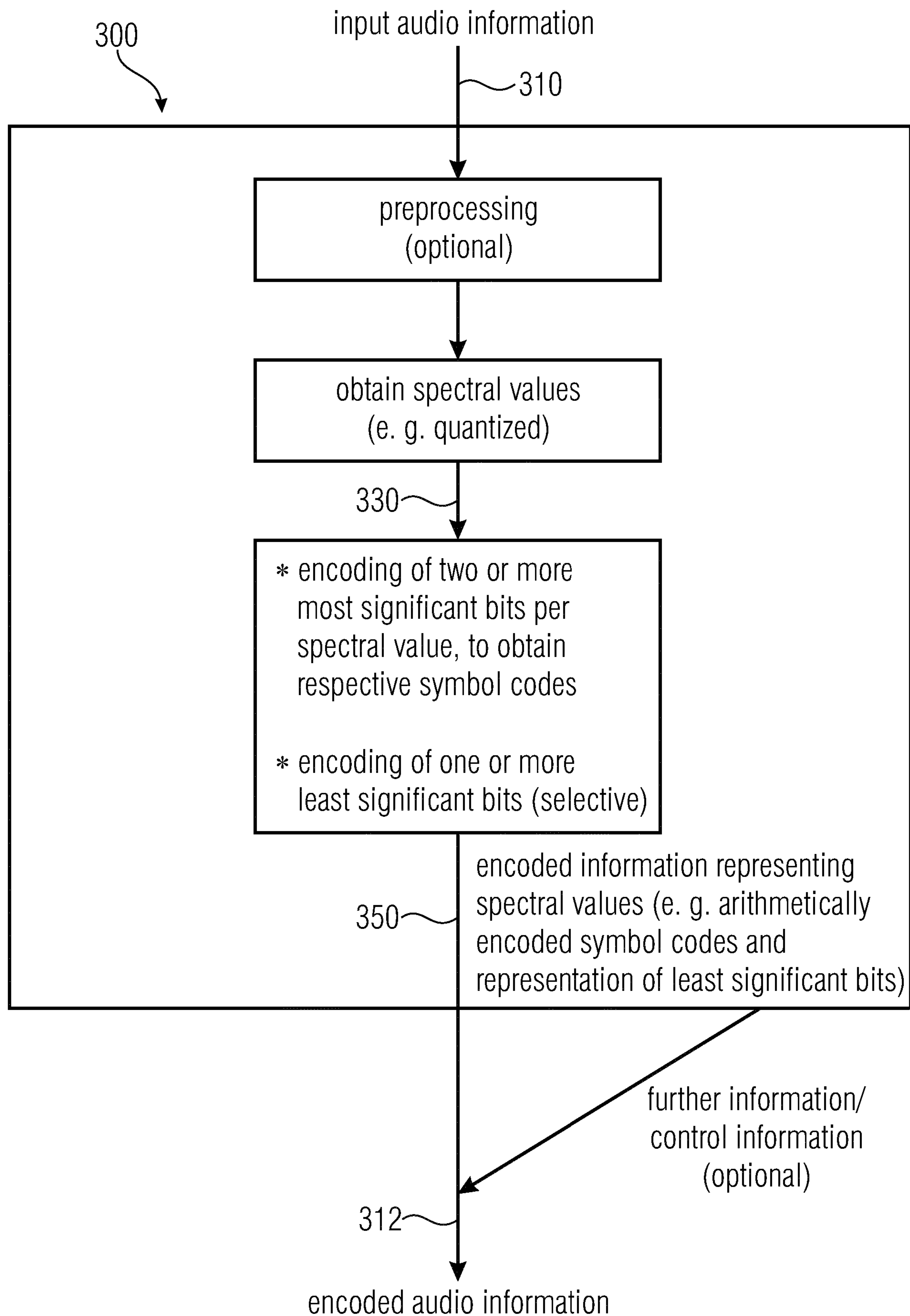


Fig. 3

4/23

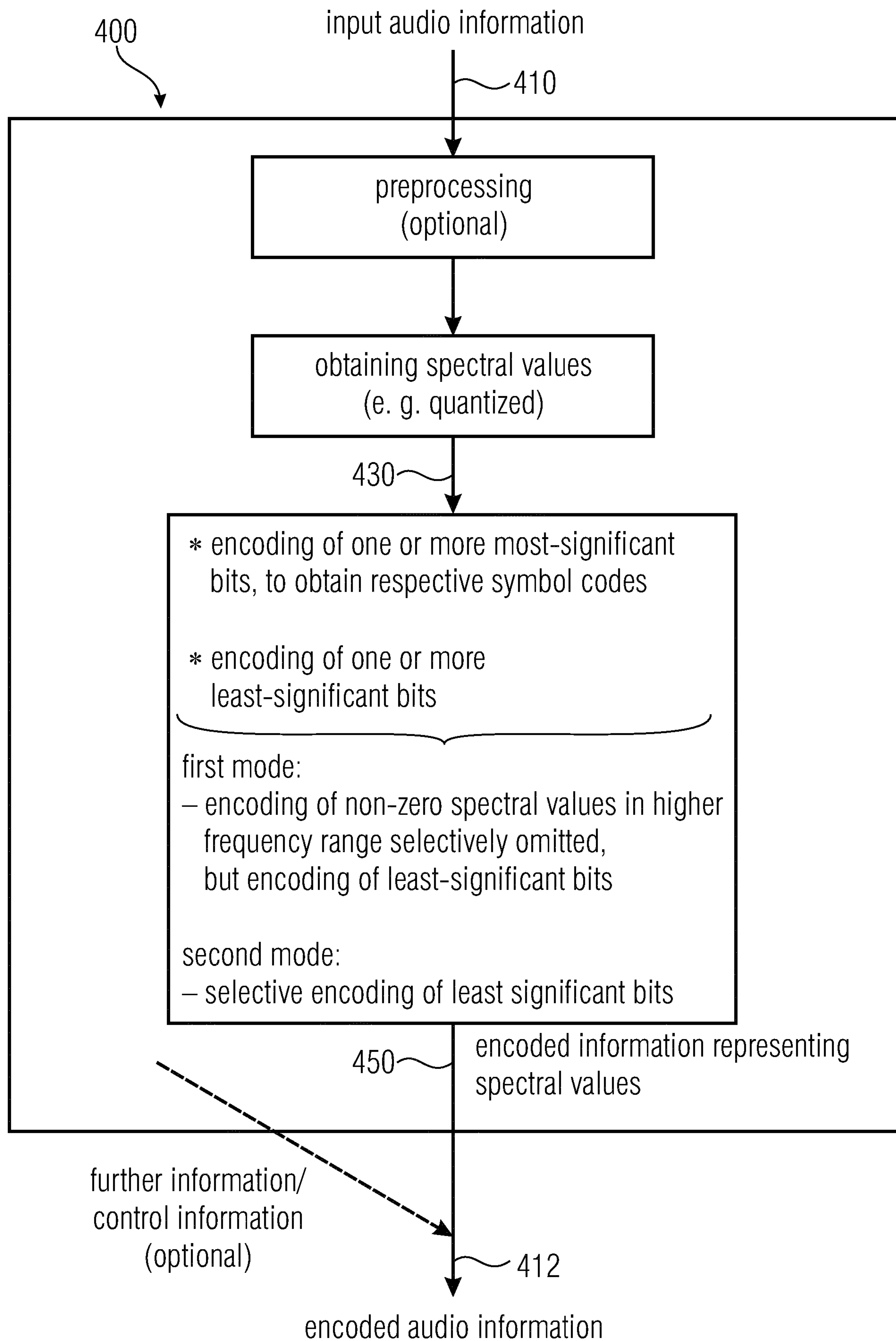


Fig. 4

5/23

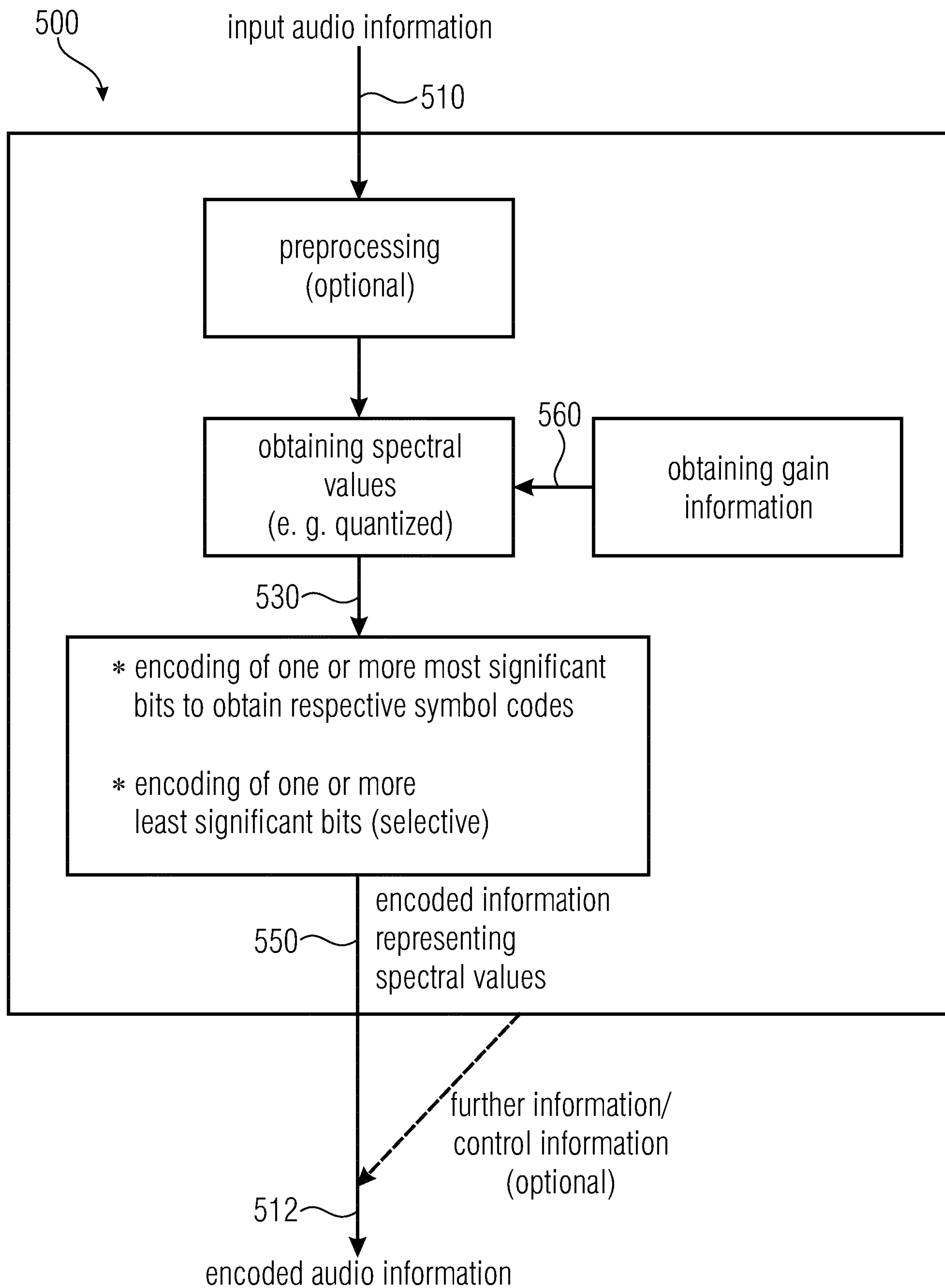


Fig. 5

6/23

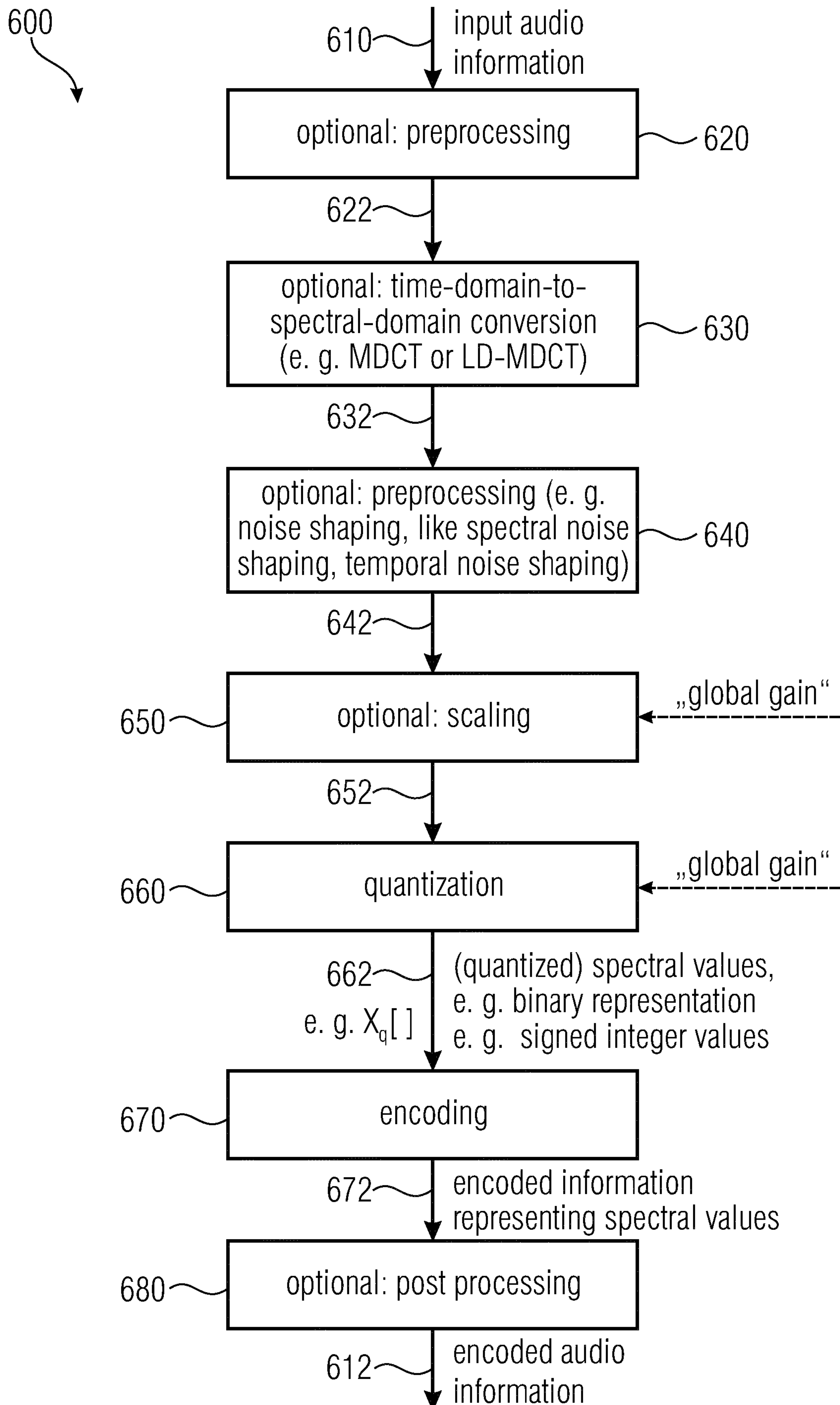


Fig. 6

7/23

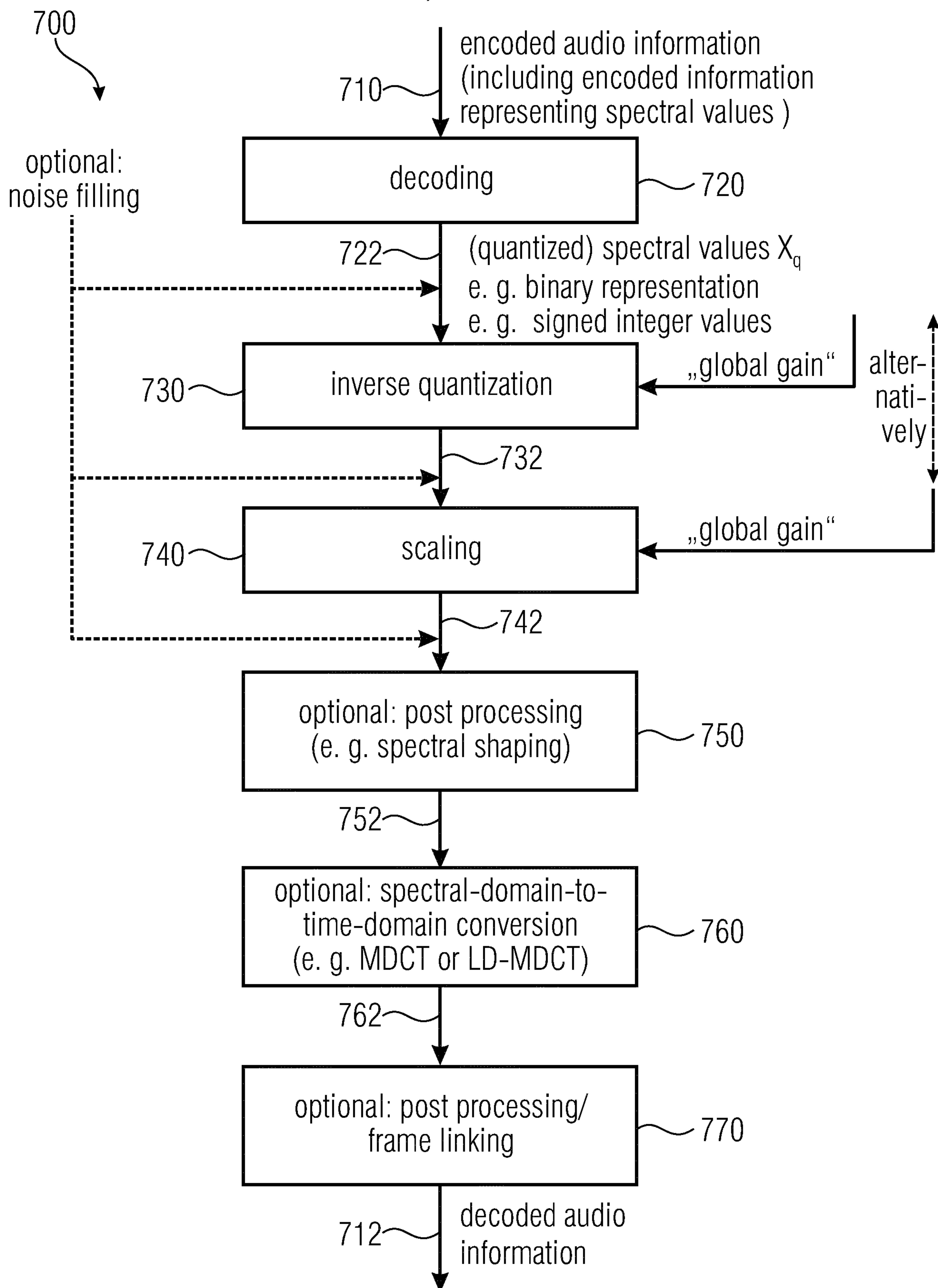


Fig. 7

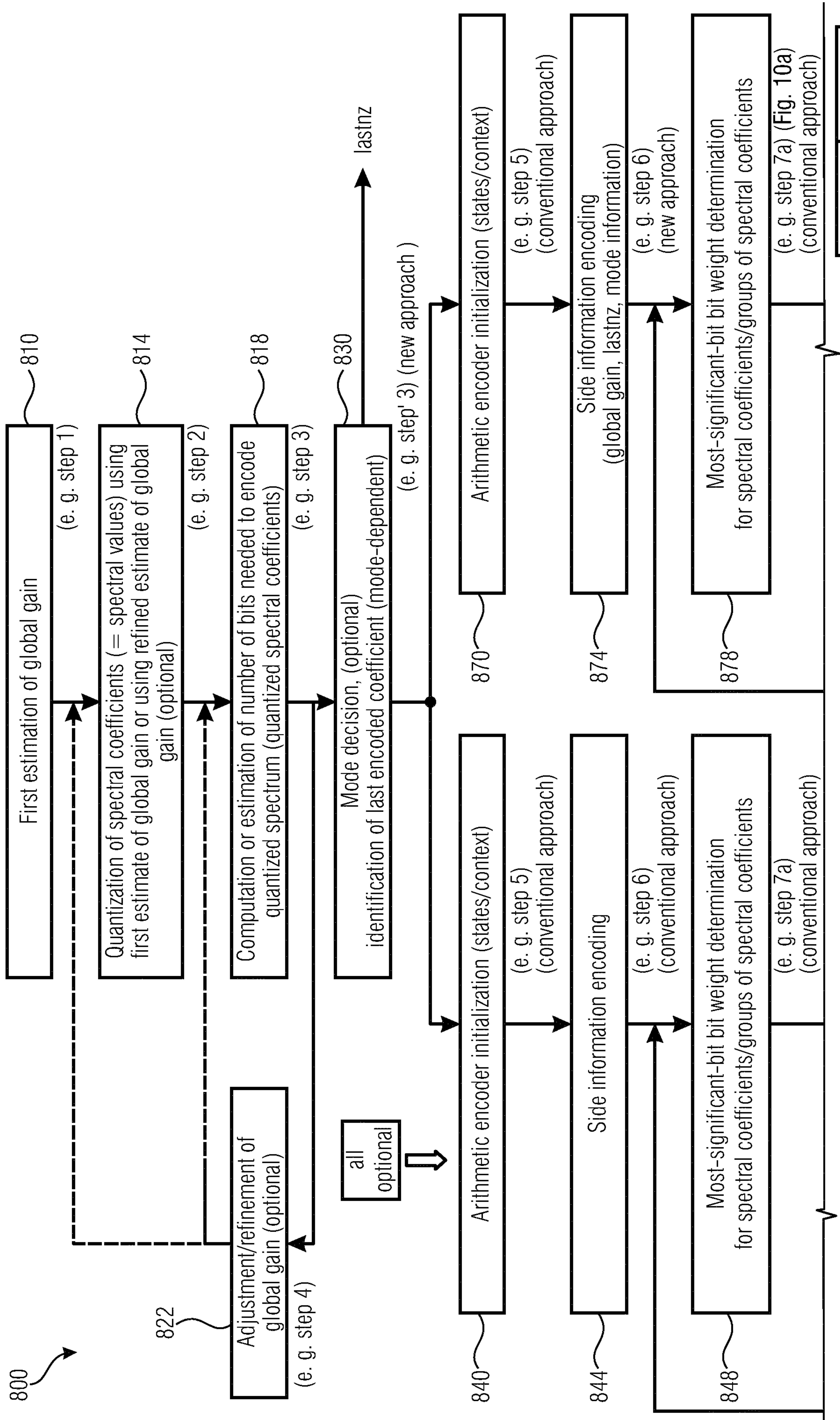


FIG 8A

FIG 8	FIG 8A
	FIG 8B

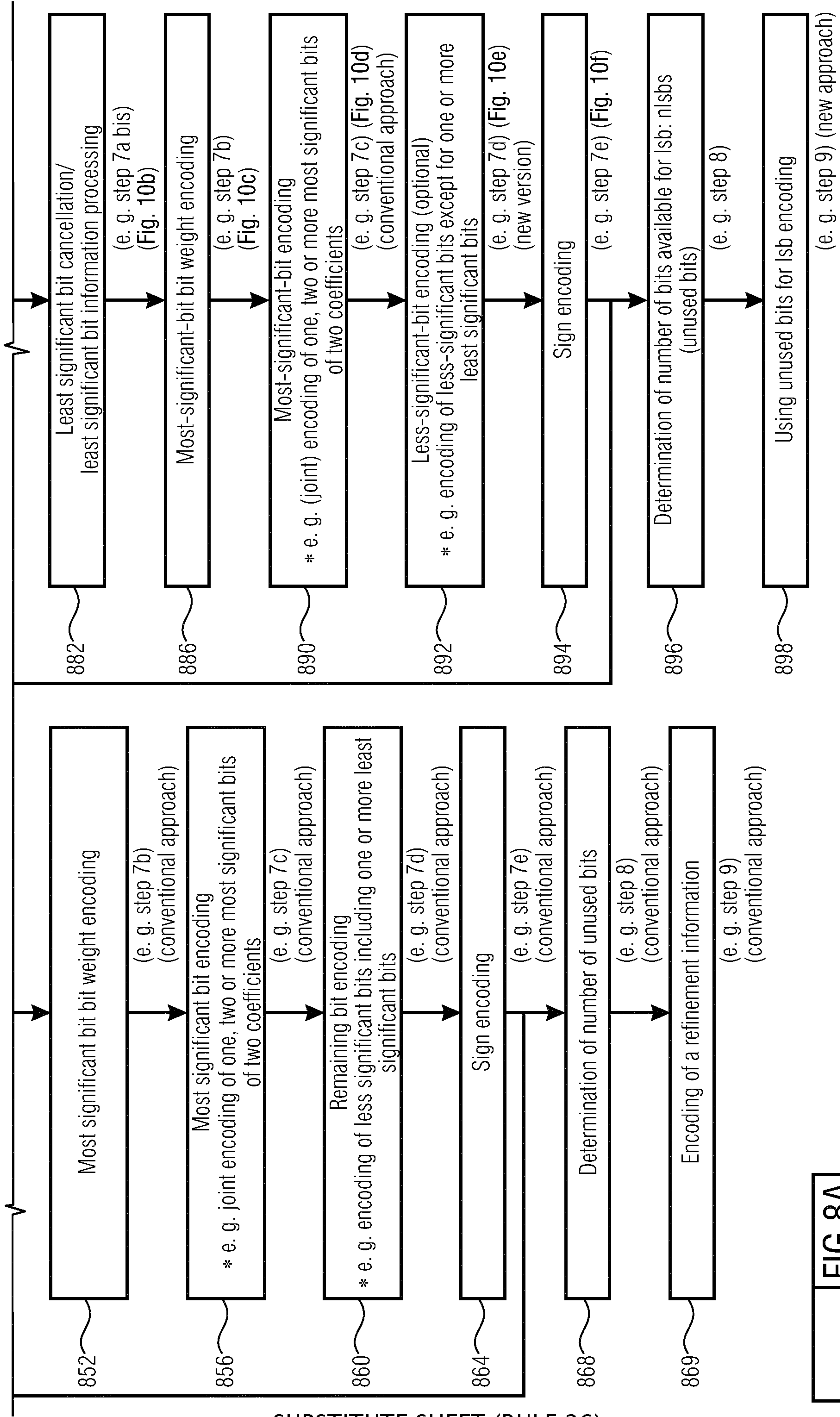
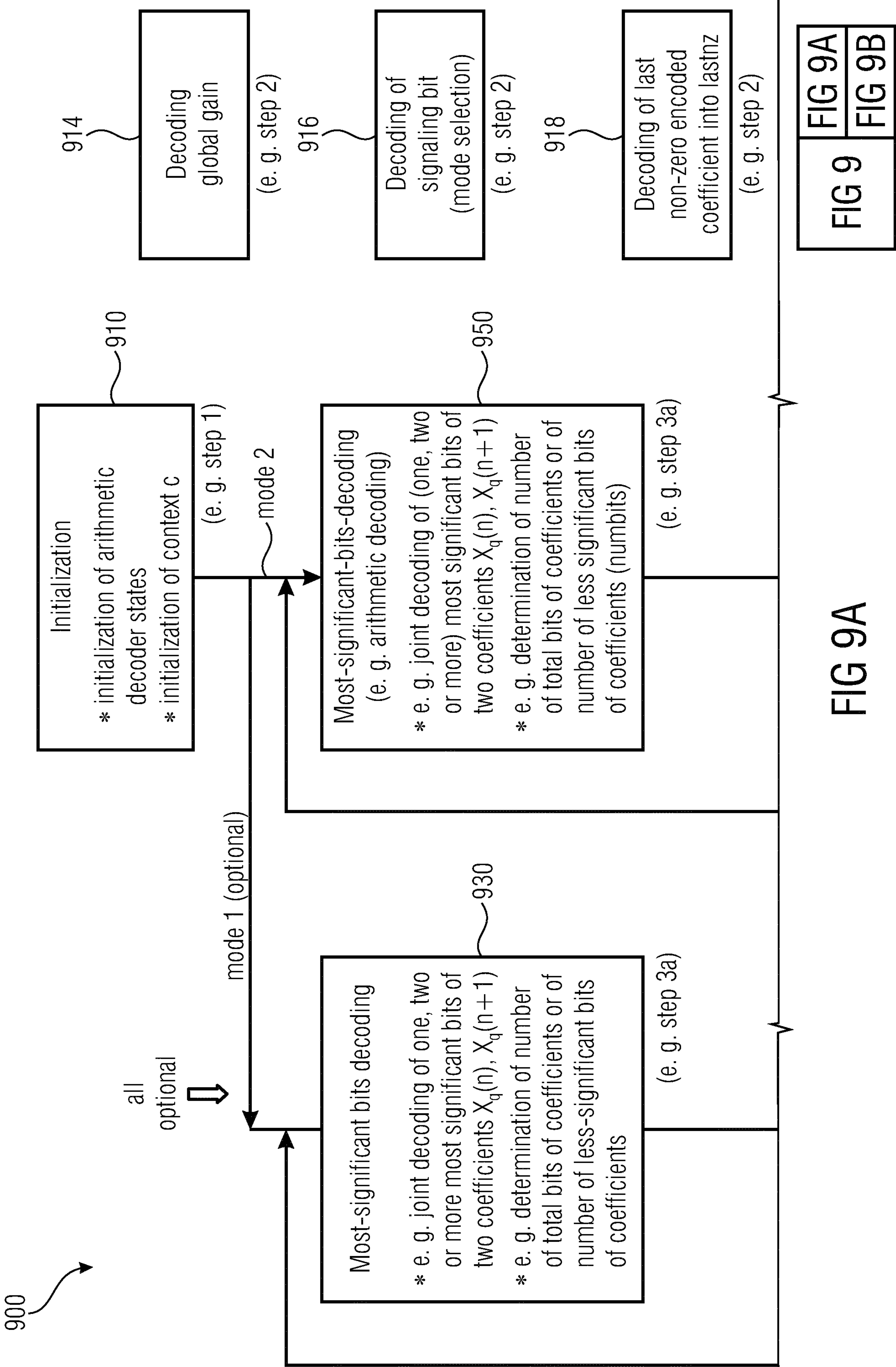


FIG 8B

FIG 8	FIG 8A
	FIG 8B



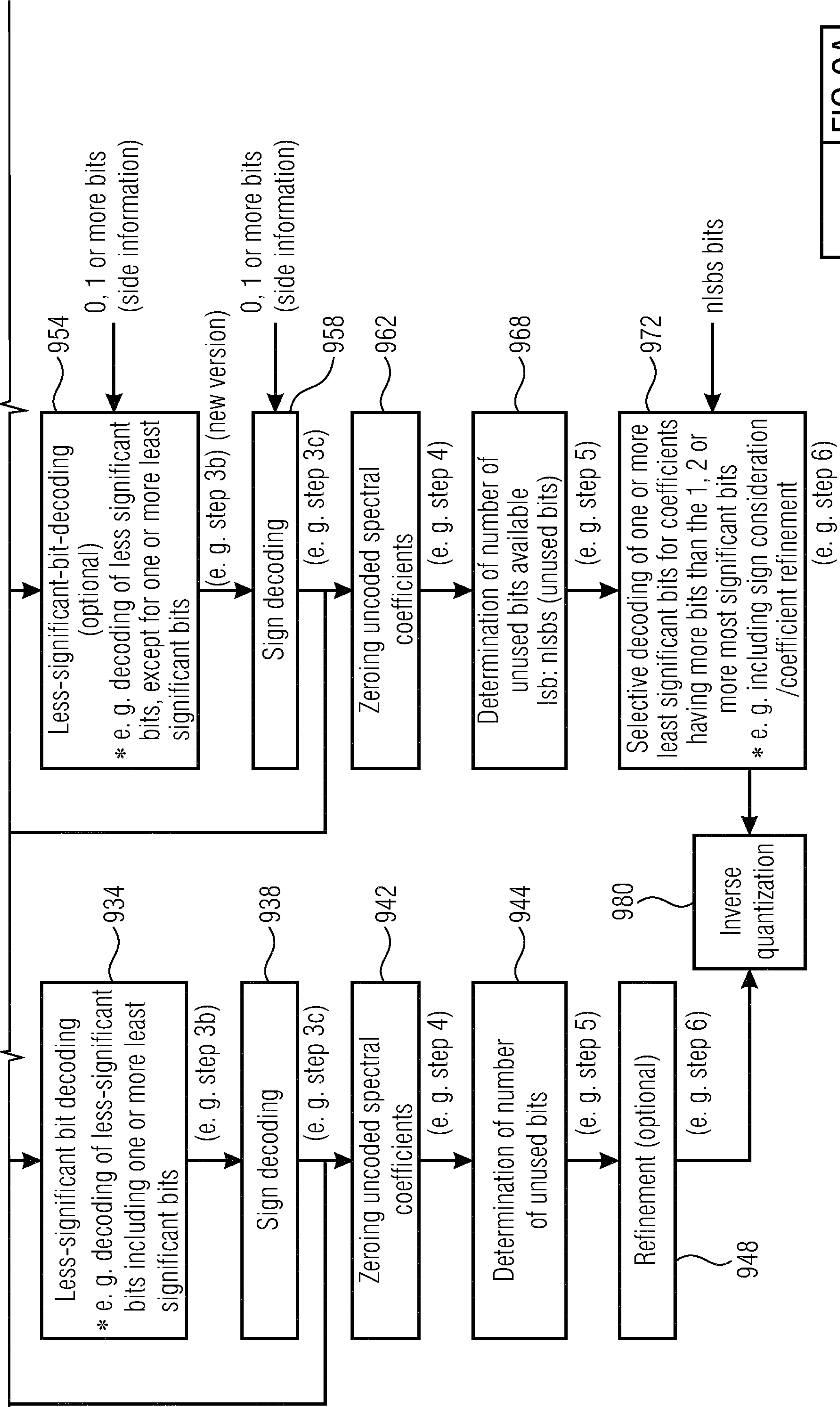


FIG 9B

FIG 9	FIG 9A
	FIG 9B

12/23

- Step 7a: compute the minimum number of bits needed to represent the amplitude of the two coefficients $X_q(n)$ and $X_q(n+1)$

numbits = ceil(log2(max(abs($X_q[n]$), abs($X_q[n+1]$)) + 1)); ↖ 1000a

Fig. 10a

- Step 7a-bis: this is an additional step just after Step 7a, which is performed if *numbits* > 2. The least significant bit of each coefficient is saved, and the coefficient is then modified such that its LSB is now zero. The sign of the coefficient is also saved in case the coefficient was originally non-zero and becomes zero after setting its LSB to zero.

```

if (numbits > 2)
{
    bit = abs( $X_q[n]$ ) & 1;                                ↖ 1010a
    lsbs[nlsbs++] = bit;                                   ↖ 1010b
    if (bit != 0 && (abs( $X_q[n]$ ) & FFFE) == 0)             ↖ 1010c
    {
        bit = 0;                                          } ↖ 1010d
        if ( $X_q[n]$  < 0) {
            bit = 1;
        }
        lsbs[nlsbs++] = bit;                               ↖ 1010e
    }
     $X_q[n] = (X_q[n]/2)*2$ ;                                ↖ 1010f
    bit = abs( $X_q[n+1]$ ) & 1;                                ↖ 1011a
    lsbs[nlsbs++] = bit;                                   ↖ 1011b
    if (bit != 0 && (abs( $X_q[n+1]$ ) & FFFE) == 0)           ↖ 1011c
    {
        bit = 0;                                          } ↖ 1011d
        if ( $X_q[n+1]$  < 0) {
            bit = 1;
        }
        lsbs[nlsbs++] = bit;                               ↖ 1011e
    }
     $X_q[n+1] = (X_q[n+1]/2)*2$ ;                             ↖ 1011f
}

```

Fig. 10b

13/23

- Step 7b: encode $numbits-2$ escape values ($VAL_ESC=16$) if $numbits > 2$

```

for (b = 0; b < numbits-2; b++) {
    Get probabilities p from context c
    Encode escape symbol VAL_ESC with ari. enc. and probabilities p
    Update context c
}

```

1020a

Fig. 10c

- Step 7c: encode the 2 most significant bits of both coefficients $X_q(n)$ and $X_q(n+1)$ as a single symbol sym (whose value lies between 0 and 15)

```

s = max(0, numbits-2);
a = abs( $X_q[n]$ ) >> s;
b = abs( $X_q[n+1]$ ) >> s;
sym = a + 4*b;
Get probabilities p from context c
Encode symbol sym with ari. enc. and probabilities p
Update context c

```

1040a

1040b

1040c

1040d

Fig. 10d

- Step 7d: encode the remaining bits except the least significant bit

```

for (b = 1; b < numbits-2; b++) {
    bit0 = (abs( $X_q[n]$ ) >> b) & 1;
    Encode bit0 as side-information
    bit1 = (abs( $X_q[n+1]$ ) >> b) & 1;
    Encode bit1 as side-information
}

```

1050a

1050b

1050c

Fig. 10e

14/23

- Step 7e: encode the sign of each coefficient, except if the coefficient is zero

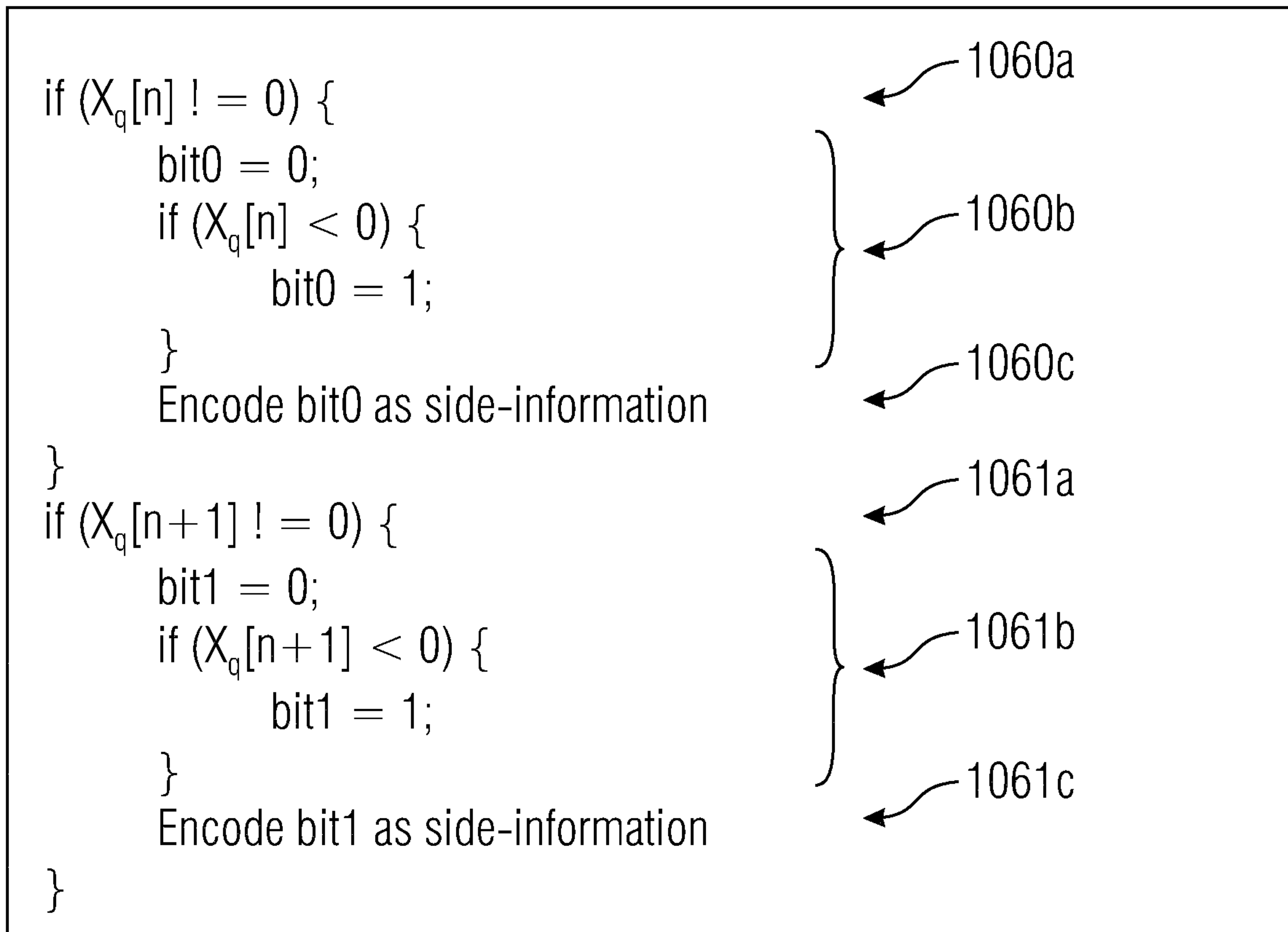


Fig. 10f

- Step 3a: decode the 2 most significant bits of both coefficients $X_q(n)$ and $X_q(n+1)$

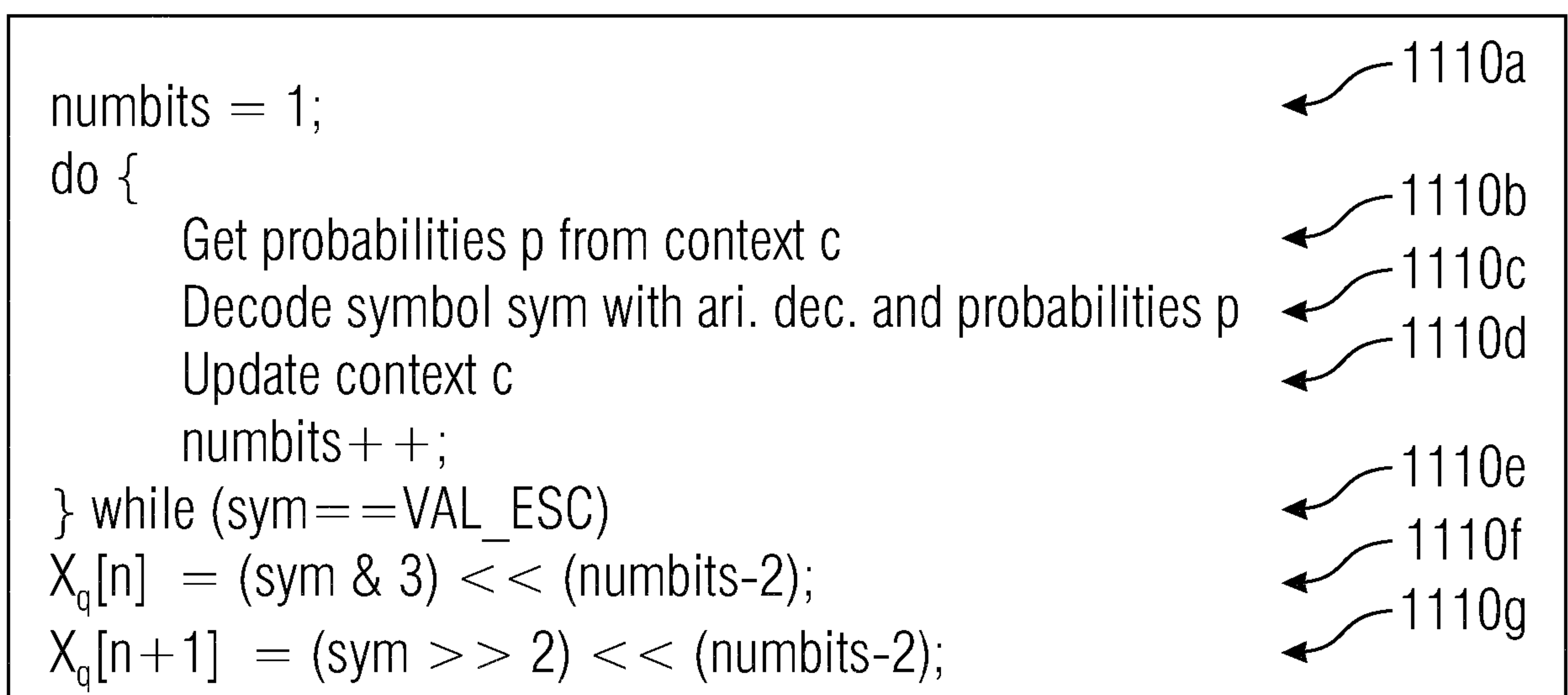


Fig. 11a

15/23

- Step 3b: decode the remaining bits except the least significant bit

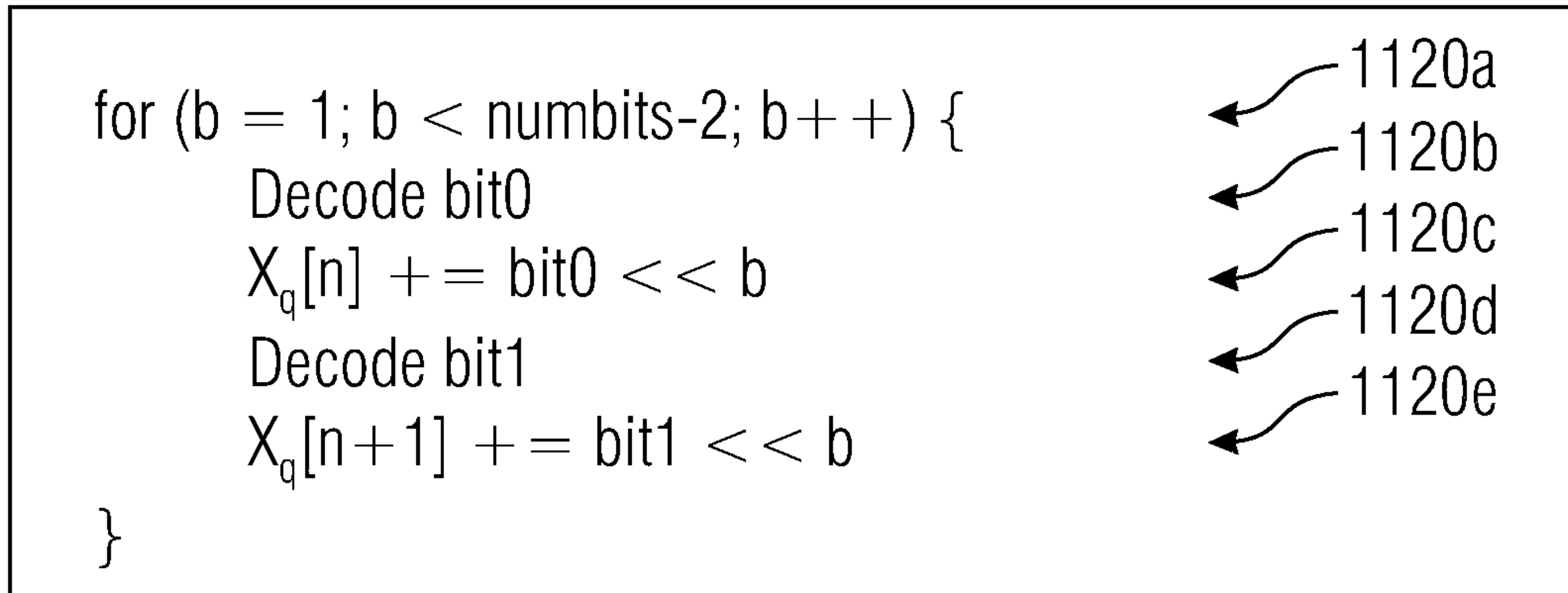


Fig. 11b

- Step 3c: decode the sign of each coefficient, except if the coefficient is zero

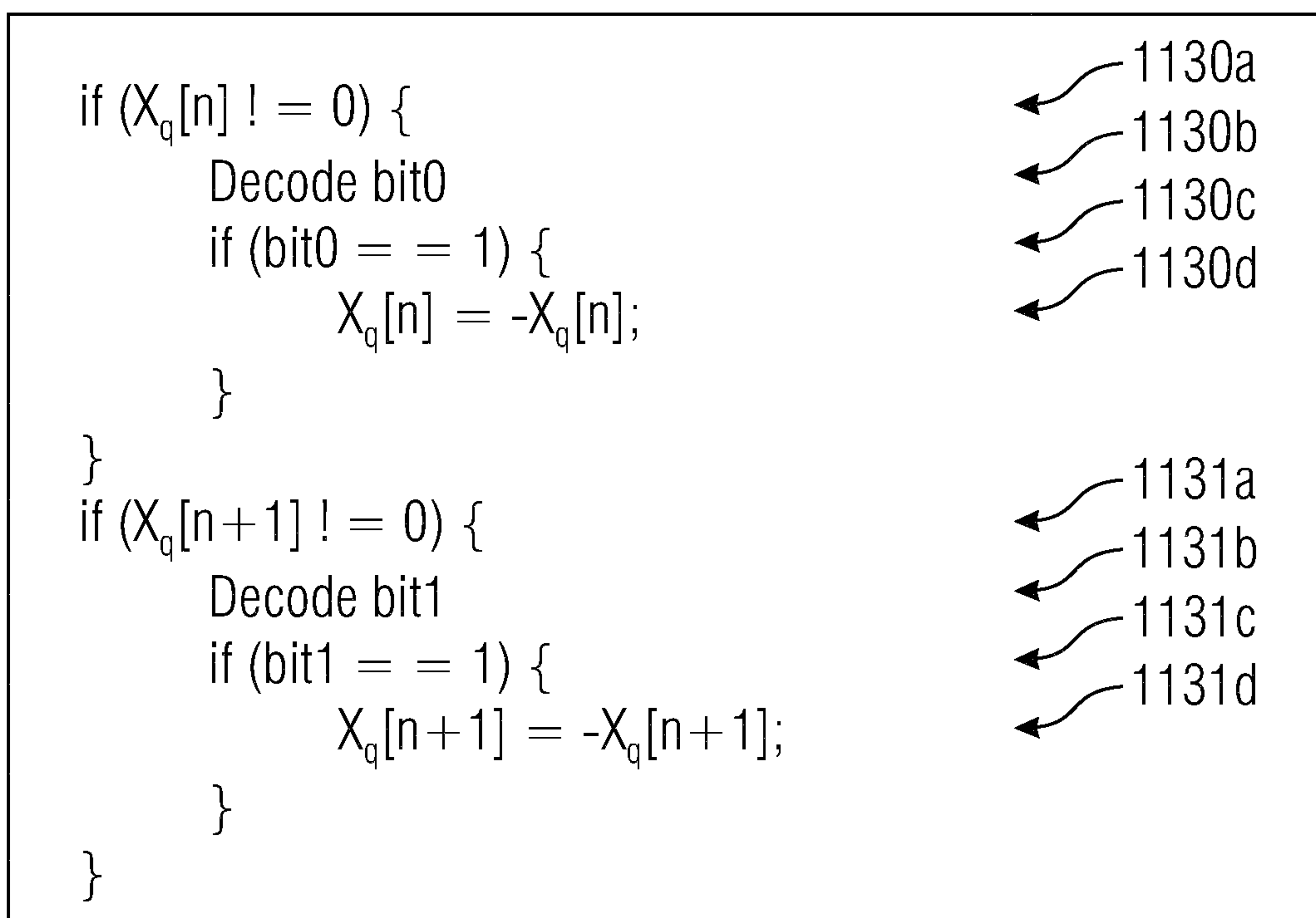


Fig. 11c

16/23

- Step 6: if there are unused bits, decode n/sbs bits and store them in $/sbs/$. Then refine the coefficients $(n, n+1)$ if $numbits[n] > 2$ using the decoded LSB bits

```

k = 0;
for (n = 0; n < lastnz; n += 2) {
    if (numbits[n] > 2) {
        if (k == nlsbs) {
            break;
        }
        bit0 = lsbs[k++];
        if (bit0 == 1) {
            if (Xq[n] > 0) {
                Xq[n] += 1;
            } else if (Xq[n] < 0) {
                Xq[n] -= 1;
            } else {
                if (k == nlsbs) {
                    break;
                }
                bit1 = lsbs[k++];
                Xq[n] = 1 - 2*bit1;
            }
        }
        if (k == nlsbs) {
            break;
        }
        bit0 = lsbs[k++];
        if (bit0 == 1) {
            if (Xq[n+1] > 0) {
                Xq[n+1] += 1;
            } else if (Xq[n+1] < 0) {
                Xq[n+1] -= 1;
            } else {
                if (k == nlsbs) {
                    break;
                }
                bit1 = lsbs[k++];
                Xq[n+1] = 1 - 2*bit1;
            }
        }
    }
}

```

Fig. 11d

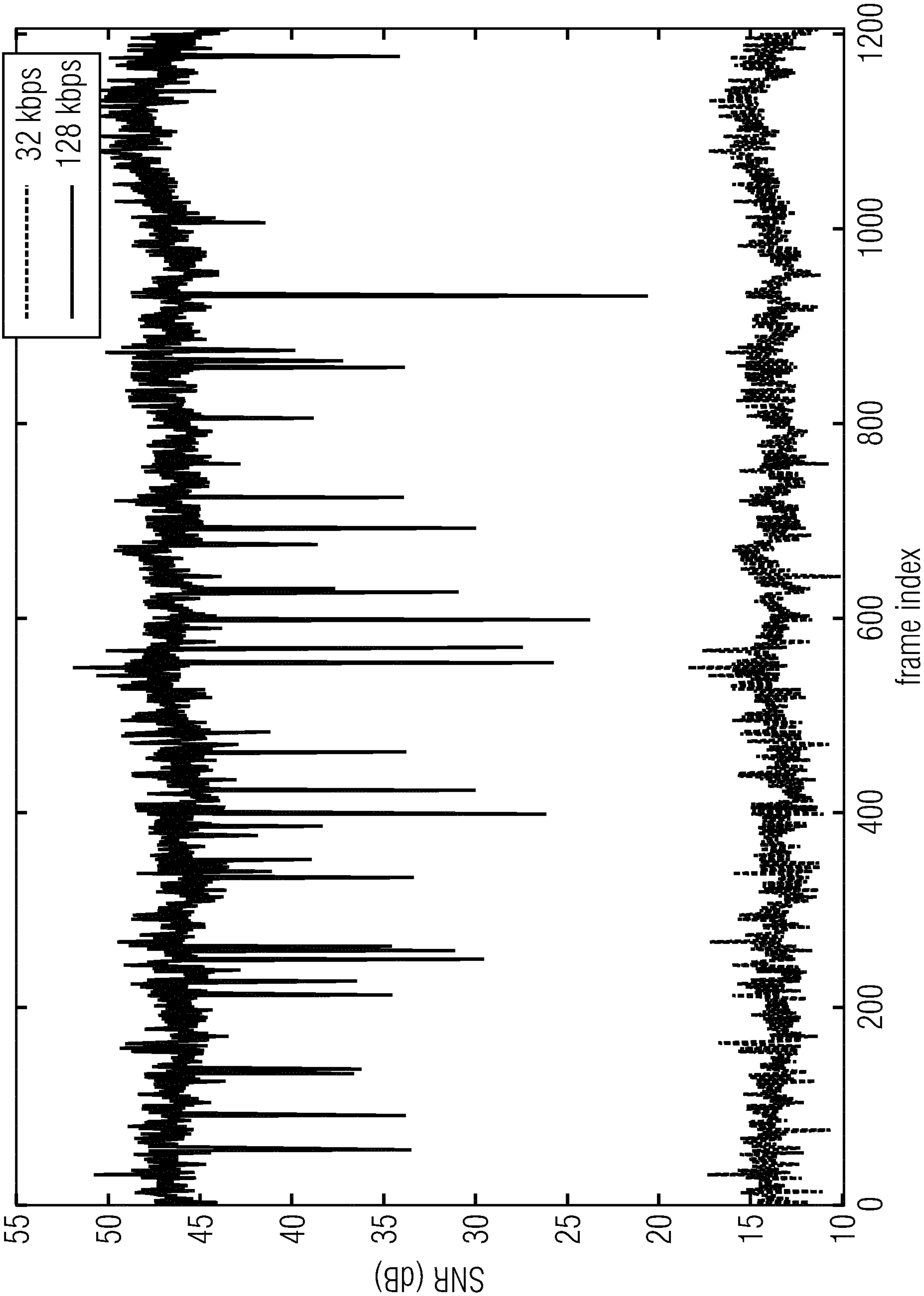


Fig. 12

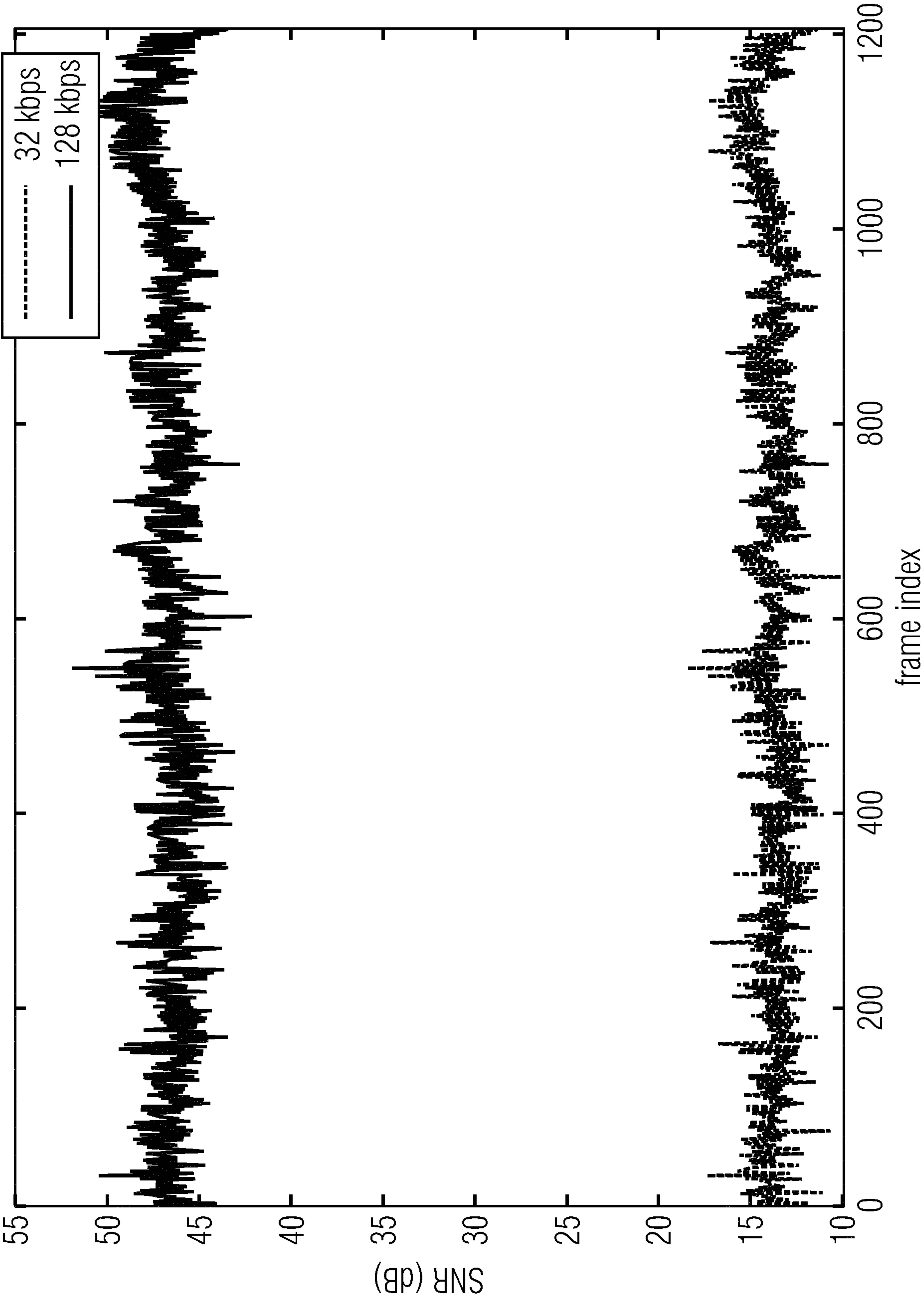


Fig. 13

19/23

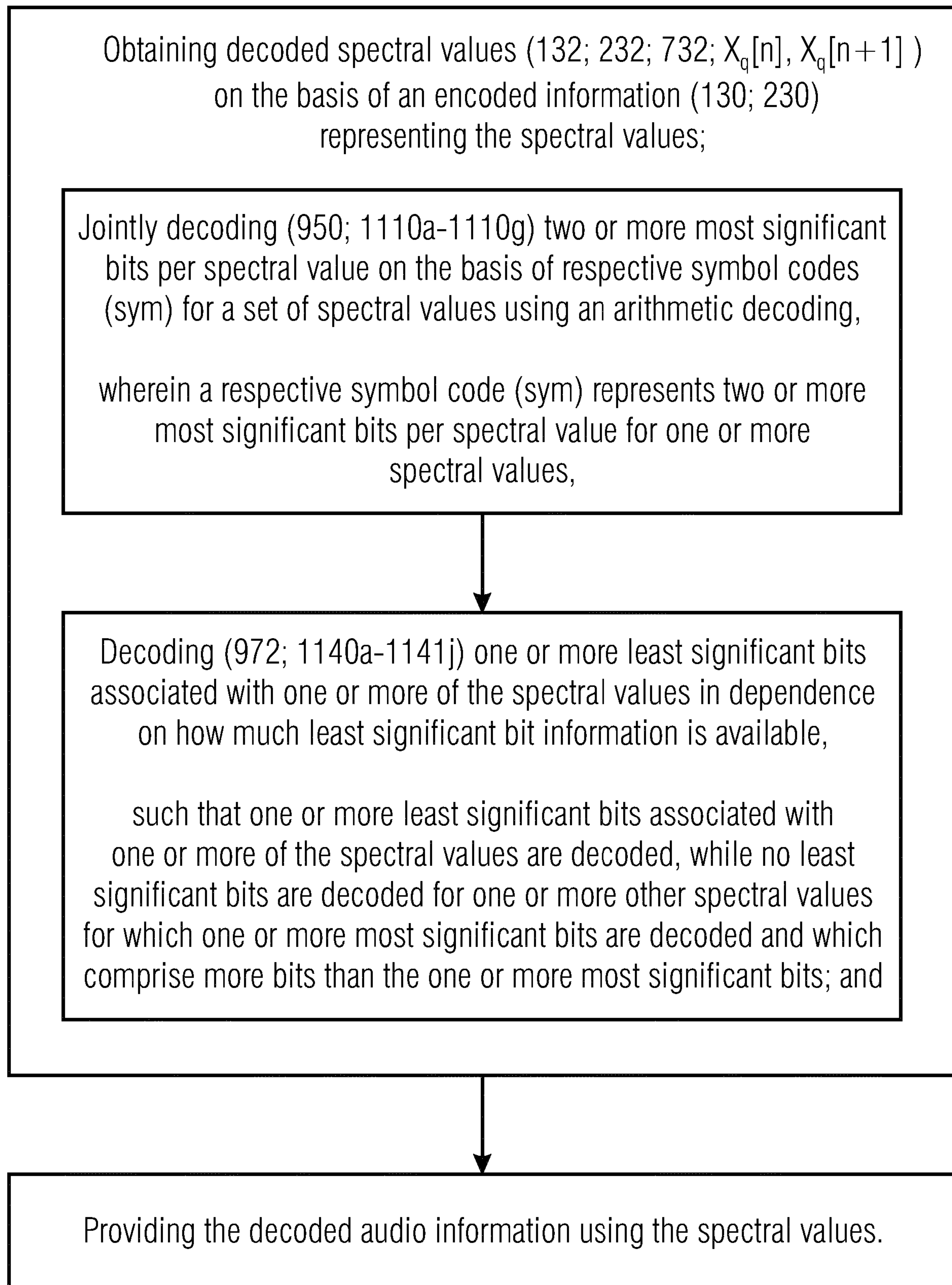


Fig. 14

Obtaining decoded spectral values (132; 232; 732; $X_q[n]$, $X_q[n+1]$) on the basis of an encoded information (130; 230) representing the spectral values,

Selecting between

- a first mode (930, 934, 938, 942, 944, 948) in which a decoding of spectral values in a higher frequency range is omitted in response to a signaling from the encoder and in which least significant bits are decoded (934) for all spectral values for which one or more most significant bits are decoded and which comprise more bits than the most significant bits, and
- a second mode (950, 954, 958, 962, 968, 972) in which one or more least significant bits associated with one or more of the spectral values are decoded, while no least significant bits are decoded for one or more other spectral values for which one or more most significant bits have been decoded and which comprise more bits than the most significant bits; and

Decoding (950; 1110a-1110g) one or more most significant bits on the basis of respective symbol codes (sym) for a plurality of spectral values ($X_q[0] \dots X_q[\text{lastnz}-1]$),

Decoding one or more least significant bits for one or more of the spectral values

Providing the decoded audio information using the spectral values.

Fig. 15

21/23

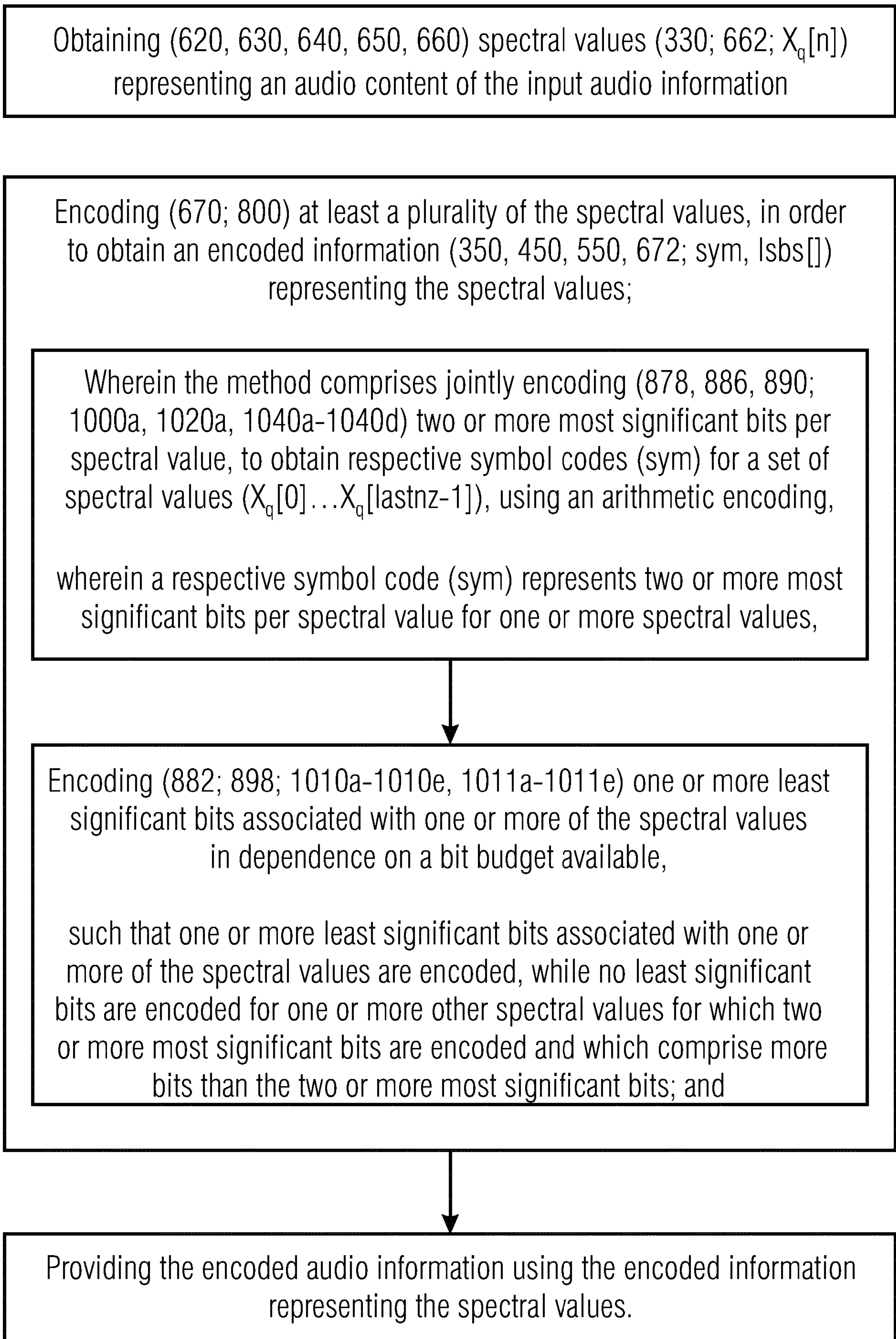


Fig. 16

22/23

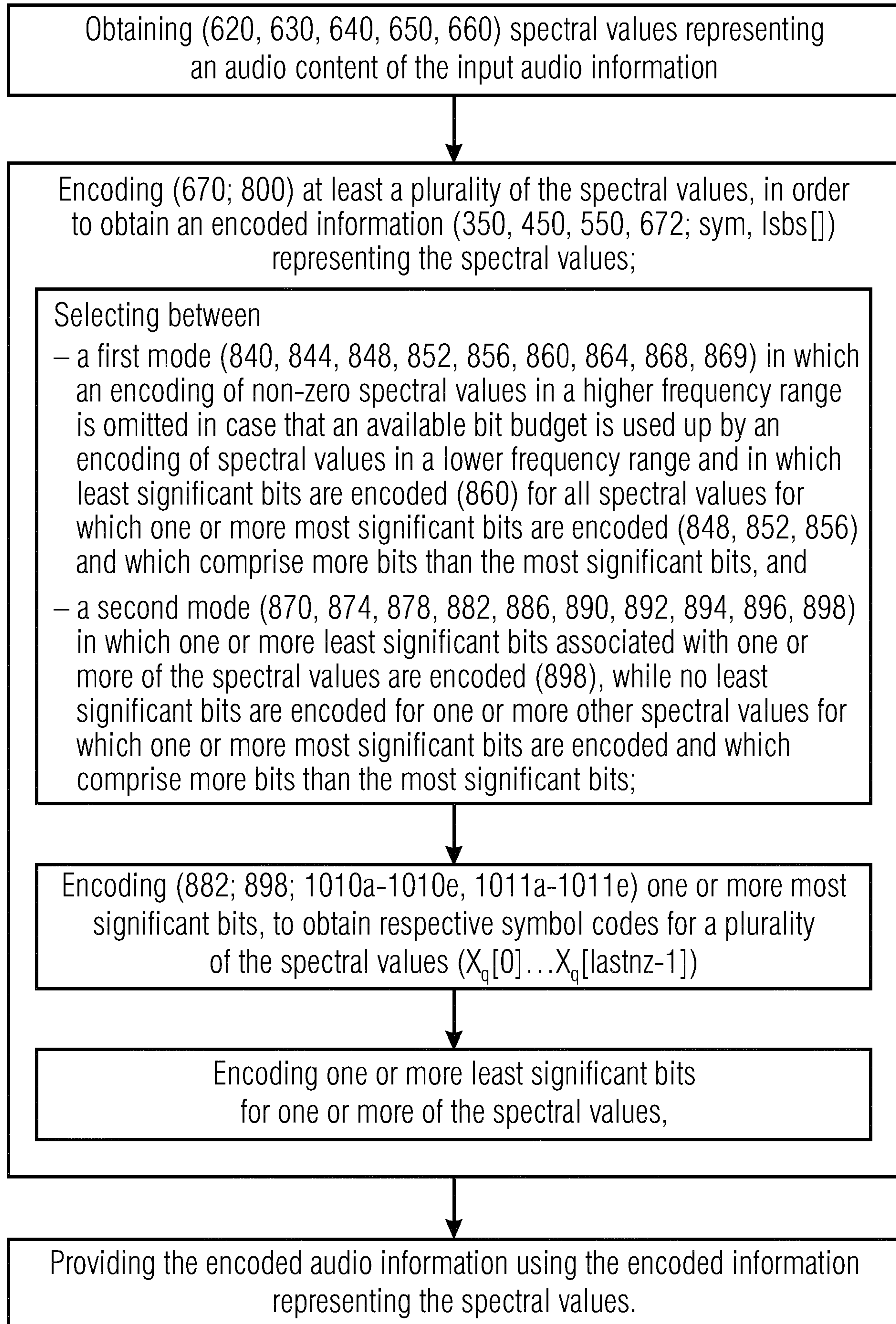


Fig. 17

23/23

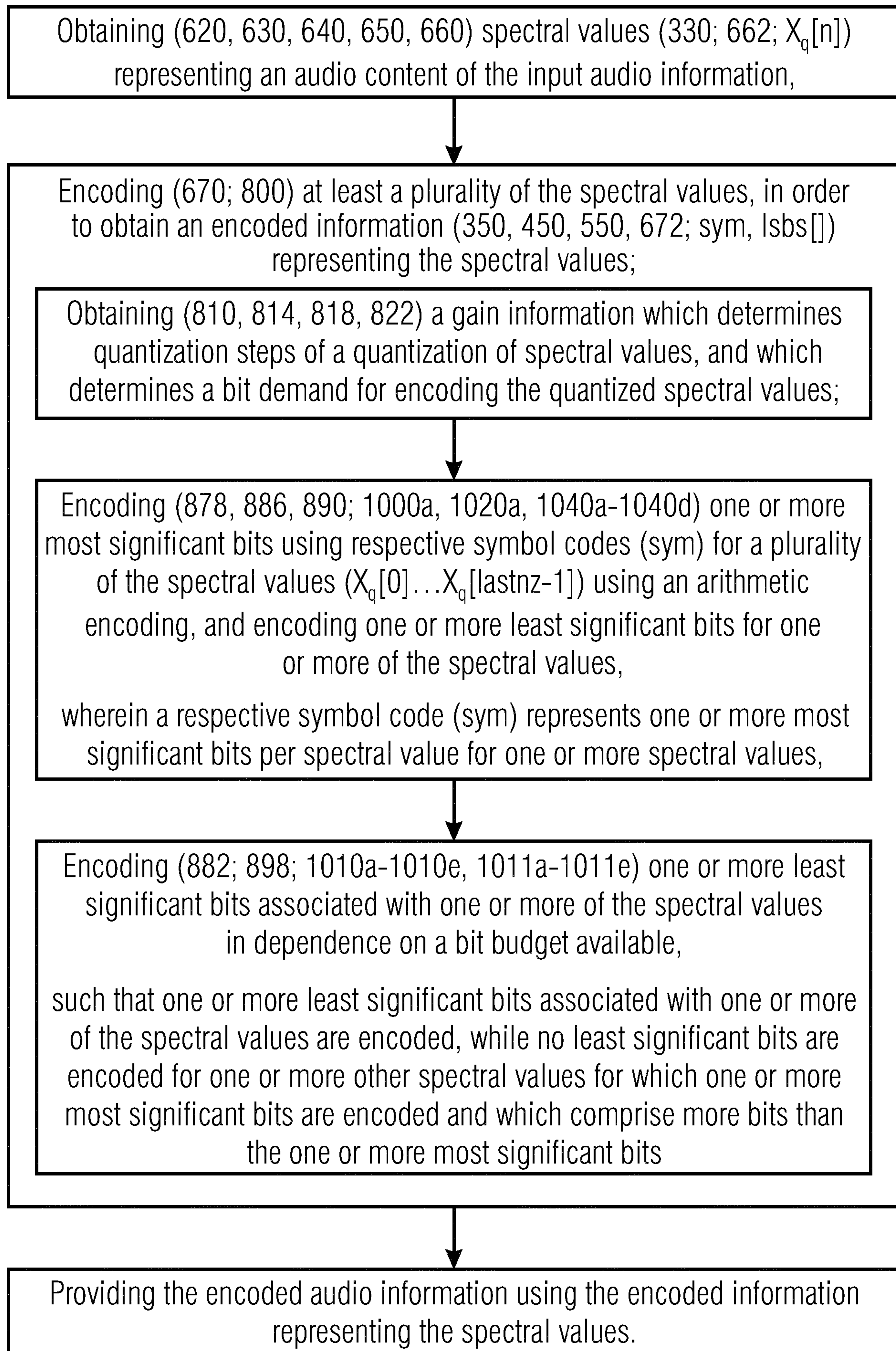


Fig. 18