

1/34

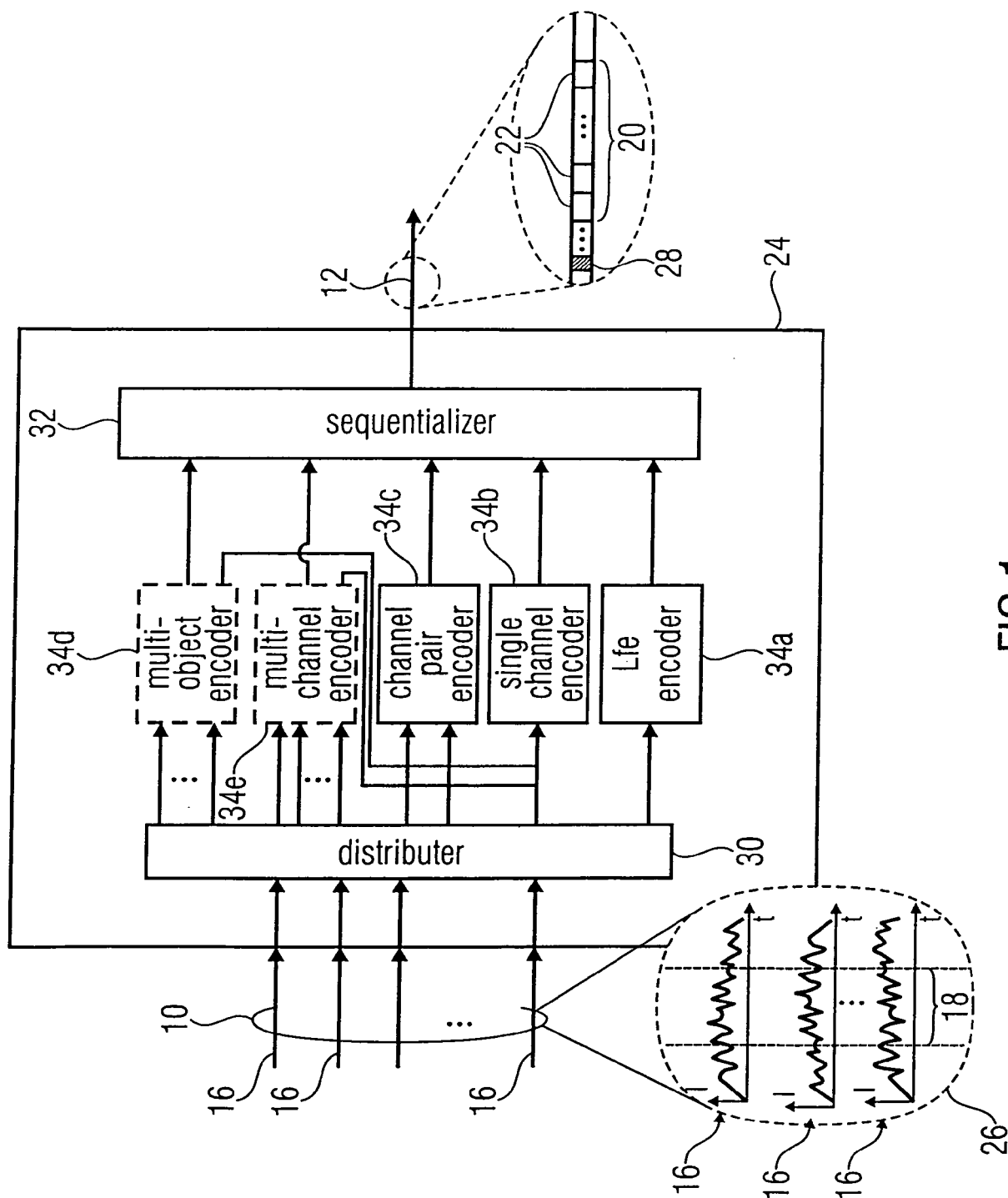


FIG 1

2/34

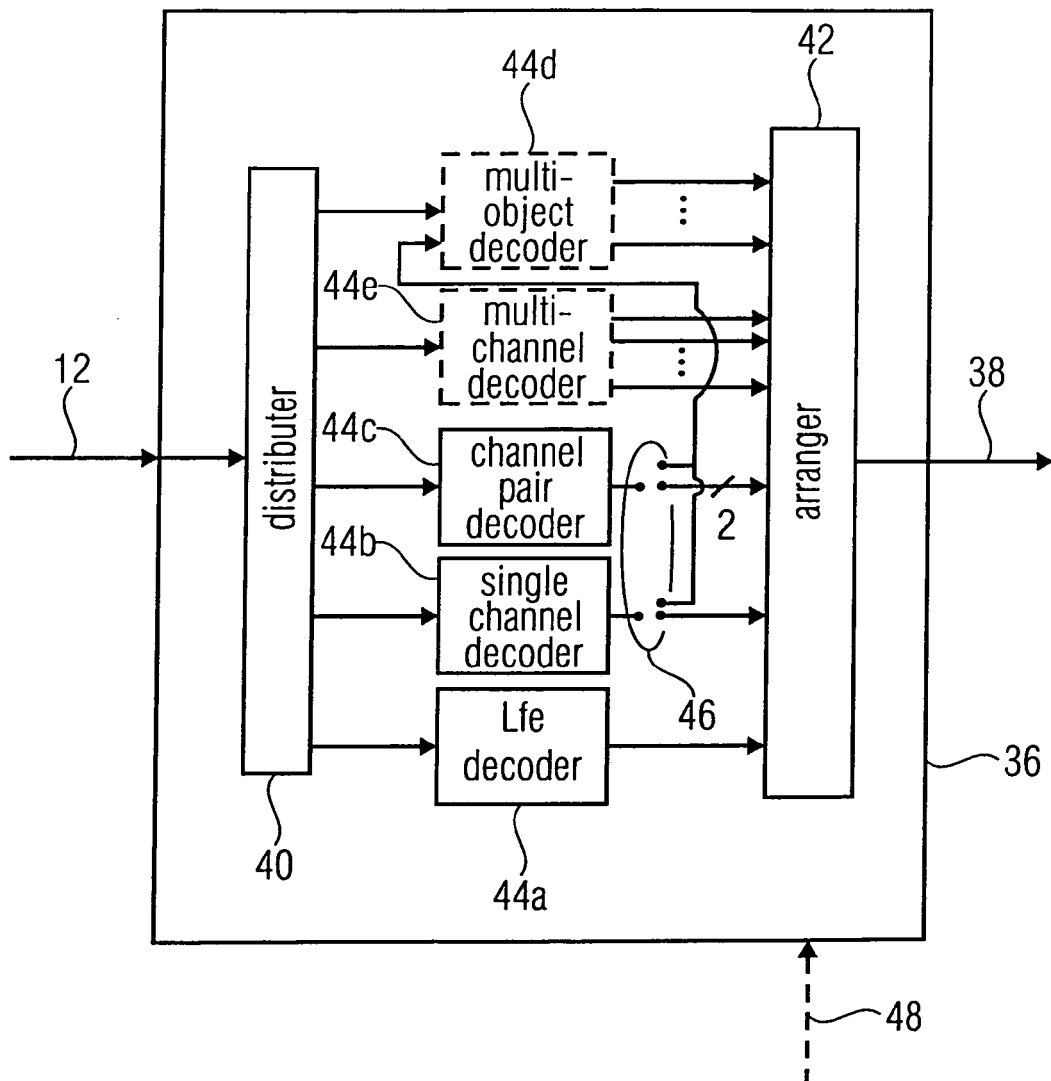


FIG 2

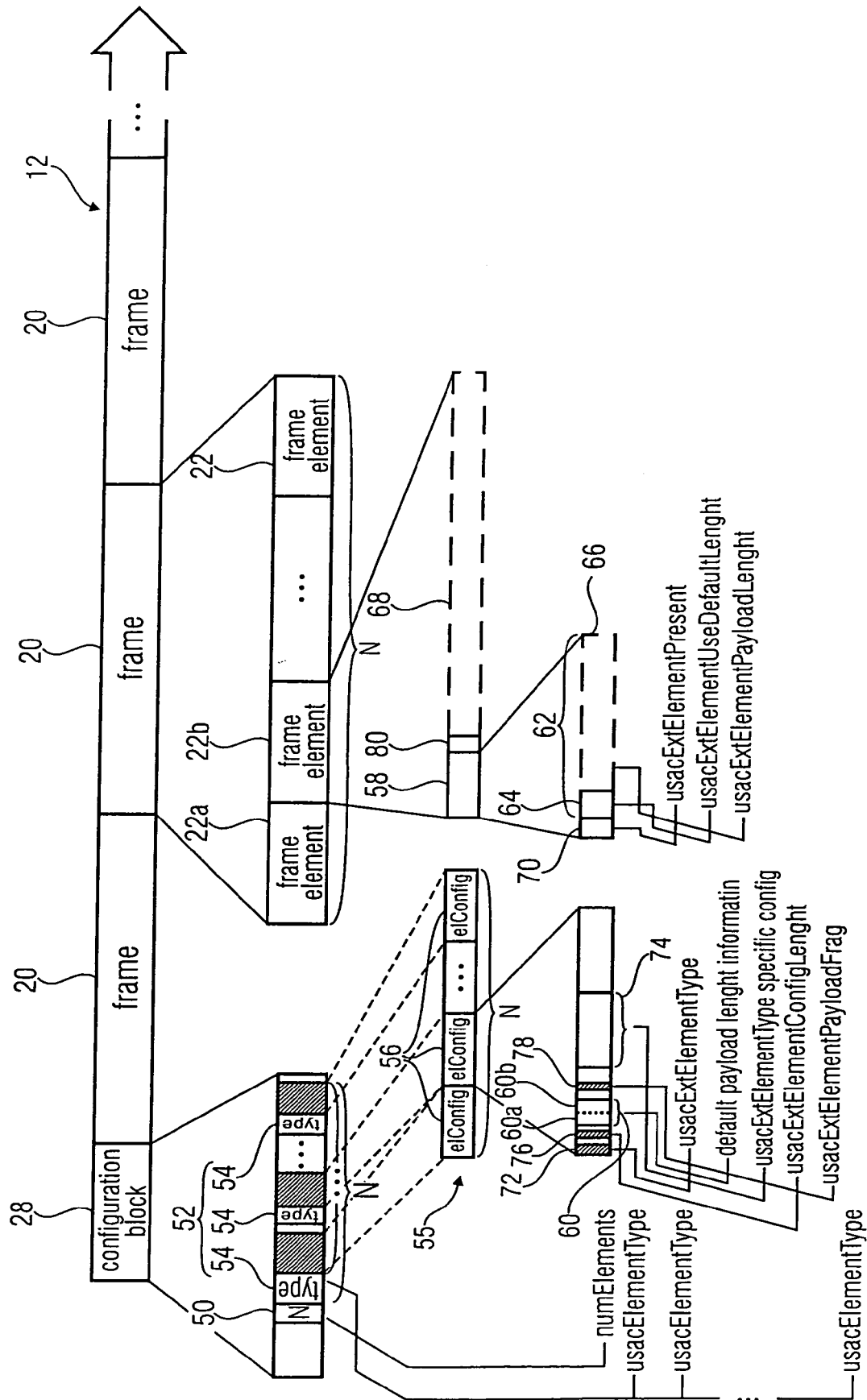


FIG 3

4/34

Syntax of UsacConfig()

Syntax	No. of bits	Mnemonic
UsacConfig() { usacSamplingFrequencyIndex ; if (usacSamplingFrequencyIndex == 0x1f) { usacSamplingFrequency ; } coreSbrFrameLengthIndex ; channelConfigurationIndex ; if (channelConfigurationIndex == 0) { UsacChannelConfig(); } UsacDecoderConfig(); if (usacConfigExtensionPresent == 1){ UsacConfigExtension(); } }	5 24 3 5 1	bslbf uimsbf uimsbf uimsbf uimsbf

FIG 4A

Syntax of UsacChannelConfig()

Syntax	No. of bits	Mnemonic
UsacChannelConfig() { numOutChannels = escapedValue(5,8,16); for (i=0; i<numOutChannels; i++) { bsOutputChannelPos[i] ; } }	5	uimsbf

FIG 4B

5/34

Syntax of UsacDecoderConfig()

Syntax	No. of bits	Mnemonic
<pre> UsacDecoderConfig() { numElements = escapedValue(4,8,16) + 1; for (elemIdx=0; elemIdx<numElements; ++elemIdx) { usacElementType[elemIdx] switch (usacElementType[elemIdx]) { case: ID_USAC_SCE UsacSingleChannelElementConfig(sbrRatioIndex); break; case: ID_USAC_CPE UsacChannelPairElementConfig(sbrRatioIndex); break; case: ID_USAC_LFE UsacLfeElementConfig(); break; case: ID_USAC_EXT UsacExtElementConfig(); break; } } } </pre>	2	uimsbf
NOTE: UsacSingleChannelElementConfig(), UsacChannelPairElementConfig(), UsacLfeElement-Config() and UsacExtElementConfig() signaled at position elemIdx refer to the corresponding elements in UsacFrame() at the respective position elemIdx.		

FIG 4C

6/34

Syntax of UsacSingleChannelElementConfig()

Syntax	No. of bits	Mnemonic
<pre> UsacSingleChannelElementConfig(sbrRatioIndex) { UsacCoreConfig(); if (sbrRatioIndex > 0) { SbrConfig(); } } </pre>		

FIG 4D

Syntax of UsacChannelPairElementConfig()

Syntax	No. of bits	Mnemonic
<pre> UsacChannelPairElementConfig(sbrRatioIndex) { UsacCoreConfig(); if (sbrRatioIndex > 0) { SbrConfig(); stereoConfigIndex; } else { stereoConfigIndex = 0; } if (stereoConfigIndex > 0) { Mps212Config(stereoConfigIndex); } } </pre>	2	uimsbf

FIG 4E

7/34

Syntax of UsacLfeElementConfig()

Syntax	No. of bits	Mnemonic
UsacLfeElementConfig() { tw_mdct = 0; noiseFilling = 0; }		

FIG 4F

Syntax of UsacCoreConfig()

Syntax	No. of bits	Mnemonic
UsacCoreConfig() { tw_mdct; noiseFilling; }	 1 1	 bslbf bsb1f

FIG 4G

Syntax of SbrConfig()

Syntax	No. of bits	Mnemonic
SbrConfig() { harmonicsSBR; bs_interTes; bs_pvc; SbrDfltHeader(); }	 1 1 1	 bsb1f bsb1f bsb1f

FIG 4H

8/34

Syntax of SbrDfltHeader()

Syntax	No. of bits	Mnemonic
SbrDfltHeader() {		
dflt_start_freq;	4	uimsbf
dflt_stop_freq;	4	uimsbf
dflt_header_extra1;	1	uimsbf
dflt_header_extra2;	1	uimsbf
if (dflt_header_extra1 == 1) {		
dflt_freq_scale;	2	uimsbf
dflt_alter_scale;	1	uimsbf
dflt_noise_bands;	2	uimsbf
}		
if (dflt_header_extra2 == 1) {		
dflt_limiter_bands;	2	uimsbf
dflt_limiter_gains;	2	uimsbf
dflt_interpol_freq;	1	uimsbf
dflt_smoothing_mode;	1	uimsbf
}		
}		

FIG 4I

9/34

Syntax of Mps212Config()

Syntax	No. of bits	Mnemonic
Mps212Config(stereoConfigIndex)		
{		
bsFreqRes;	3	uimsbf
bsFixedGainDMX;	3	uimsbf
bsTempShapeConfig;	2	uimsbf
bsDecorrConfig;	2	uimsbf
bsHighRateMode;	1	uimsbf
bsPhaseCoding;	1	uimsbf
bsOttBandsPhasePresent;	1	uimsbf
if (bsOttBandsPhasePresent) {		NOTE 1
bsOttBandsPhase;	5	uimsbf
}		
if (bsResidualCoding) {		NOTE 2
bsResidualBands;	5	uimsbf
bsOttBandsPhase = max(bsOttBandsPhase,bsResidualBands);		
bsPseudoLr;	1	uimsbf
}		
if (bsTempShapeConfig == 2) {		
bsEnvQuantMode;	1	uimsbf
}		
}		
NOTE 1: if bsOttBandsPhasePresent == 0 bsOttBandsPhase ist initialized according to Table 104.		
NOTE 2: bsResidualCoding depends on stereoConfigIndex according to Table 72.		

FIG 4J

10/34

Syntax of UsacExtElementConfig()

Syntax	No. of bits	Mnemonic
UsacExtElementConfig() { usacExtElementType = escapedValue(4,8,16); usacExtElementConfigLenght = escapedValue(4,8,16); usacExtElementDefaultLenghtPresent; if (usacExtElementDefaultLenghtPresent) { usacExtElementDefaultLenght = escapedValue(8,16,0) + 1; } else { usacExtElementDefaultLenght = 0; } usacExtElementPayloadFrag; switch (usacExtElementType) { case ID_EXT_ELE_FILL: break; case ID_EXT_ELE_MPEGS: SpatialSpecificConfig(); break; case ID_EXT_ELE_SAOC: SaocSpecificConfig(); break; default: while (usacExtElementConfigLenght--) { tmp; } break; } } }		
	1	uimsbf
	1	uimsbf
	NOTE	
	8	uimsbf
NOTE: The default entry for the usacExtElementType is used for unknown extElementTypes so that legacy decoders can cope with future extensions.		

FIG 4K

11/34

Syntax of UsacConfigExtension()

Syntax	No. of bits	Mnemonic
<pre> UsacConfigExtension() { numConfigExtensions = escapedValue(2,4,8) + 1; for (confExtIdx=0; confExtIdx<numConfigExtensions; confExtIdx++) { usacConfigExtType[confExtIdx] = escapedValue(4,8,16); usacConfigExtLenght[confExtIdx] = escapedValue(4,8,16); switch (usacConfigExtType[confExtIdx]) { case ID_CONFIG_EXT_FILL: while (usacConfigExtLenght[confExtIdx]--) { fill_byte[i]; /* should be '10100101' */ } break; default: while (usacConfigExtLenght[confExtIdx]--) { tmp; } break; } } } </pre>	8	uimsbf
	8	uimsbf

FIG 4L

12/34

Syntax of escapedValue()

Syntax	No. of bits	Mnemonic
escapedValue(nBits1, nBits2, nBits3)		
{		
value;	nBits1	uimsbf
if (value == $2^{nBits1} - 1$) {		
value += valueAdd;	nBits2	uimsbf
if (valueAdd == $2^{nBits2} - 1$) {		
value += valueAdd;	nBits3	uimsbf
}		
}		
return value;		
}		

FIG 4M

13/34

Syntax of UsacFrame()
top level payload for radio object type USAC

Syntax	No. of bits	Mnemonic
<pre> UsacFrame() { usacIndependencyFlag; for (elemIdx=0; elemIdx<numElements; ++elemIdx) { switch (usacElementType[elemIdx]) { case: ID_USAC_SCE UsacSingleChannelElement(usacIndependencyFlag); break; case: ID_USAC_CPE UsacChannelPairElement(usacIndependencyFlag); break; case: ID_USAC_LFE UsacLfeElement(usacIndependencyFlag); break; case: ID_USAC_EXT UsacExtElement(usacIndependencyFlag); break; } } } </pre>	1	uimsbf

FIG 4N

14/34

Syntax of UsacSingleChannelElement()

Syntax	No. of bits	Mnemonic
<pre>UsacSingle ChannelElement(indepFlag) { UsacCoreCoderData(1. indepFlag); if (sbrRatioIndex > 0) { UsacSbrData(1. indepFlag); } }</pre>		

FIG 40

15/34

Syntax of UsacChannelPairElement()

Syntax	No. of bits	Mnemonic
<pre> UsacChannelPairElement(indepFlag) { if (stereoConfigIndex == 1) { nrCoreCoderChannels = 1; } else { nrCoreCoderChannels = 2; } UsacCoreCoderData(nrCoreCoderChannels, indepFlag); if (sbrRatioIndex > 0) { if (stereoConfigIndex == 0 stereoConfigIndex == 3) { nrSbrChannels = 2; } else { nrSbrChannels = 1; } UsacSbrData(nrSbrChannels, indepFlag); } if (stereoConfigIndex > 0) { Mps212Data(indepFlag); } } </pre>		

FIG 4P**Syntax of UsacLfeElement()**

Syntax	No. of bits	Mnemonic
<pre> UsacLfeElement(indepFlag) { fd_channel_stream(0,0,0,0 indepFlag); } </pre>		

FIG 4Q

16/34

Syntax of UsacExtElement()

Syntax	No. of bits	Mnemonic
<pre> UsacExtElement(indepFlag) { usacExtElementPresent if (usacExtElementPresent == 1) { usacExtElementUseDefaultLenght; if (usacExtElementUseDefaultLenght) { usacExtElementPayloadLenght = usacExtElementDefaultLenght; } else { usacExtElementPayloadLenght = escapedValue(8,16,0); } if (usacExtElementPayloadLenght > 0) { if (usacExtElementPayloadFrag) { usacExtElementStart; usacExtElementStop; } else { (usacExtElementStart = 1; (usacExtElementStop = 1; } for (i=0; i<usacExtElementPayloadLenght; i++) { usacExtElementSegmentData[i] } } } } </pre>	1 1 1 1 8	uimbsbf uimbsbf uimbsbf uimbsbf uimbsbf

FIG 4R

17/34

Syntax of UsacCoreCoderData()

Syntax	No. of bits	Mnemonic
<pre> UsacCoreCoderData(nrChannels, indepFlag) { for (ch=0; ch < nrChannels; ch++) { core_mode[ch]; } if (nrChannels == 2) { StereoCoreToolInfo(core_mode); } for (ch=0; ch < nrChannels; ch++) { if (core_mode[ch] == 1){ lpd_channel_stream(indepFlag); } else { if ((nrChannels == 1) (core_model[0] != core_model[1])) { tns_data_present[ch]; } fd_channel_stream(common_window, common_tw, tns_data_present[ch], noiseFilling, indepFlag); } } } </pre>	1 1	uimsbf uimsbf

FIG 4S

18/34

Syntax of StereoCoreToolInfo()

Syntax	No. of bits	Mnemonic
StereoCoreToolInfo(core_mode)		
{		
if (core_mode[0] == 0 && core_model[1] == 0) {		
tns_active;	1	uimsbf
common_window) {	1	uimsbf
if (common_window) {		
ics_info();		
common_max_sfb;	1	uimsbf
if (common_max_sfb == 0) {		
if (window_sequence == EIGHT_SHORT_SEQUENCE) {		
max_sfb 1;	4	uimsbf
} else {		
max_sfb 1;	6	uimsbf
}		
} else {		
max_sfb 1 = max_sfb;		
}		
max_sfb_ste = max(max_sfb, max_sfb1);		
ms_mask_present;	2	uimsbf
if (ms_mask_present == 1) {		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb; sfb++) {		
ms_used[g][sfb];	1	uimsbf
}		
}		
}		
}		
}		

FIG 4T	FIG 4T-1
	FIG 4T-2
	FIG 4T-3

FIG 4T-1

19/34

```

    if (ms_mask_present == 3) {
        cplx_pred_data();
    } else {
        alpha_q_re[g][sfb] = 0;
        alpha_q_im[g][sfb] = 0;
    }
}
if (tw_mdct) {
    common_tw;
    if (common_tw) {
        tw_data();
    }
}
if (tns_active) {
    if (common_window) {
        common_tns;
    } else {
        common_tns = 0;
    }
}
tns_on_lr
if (common_tns) {
    tns_data();
    tns_data_present[0] = 0;
    tns_data_present[1] = 0;
}

```

1 uimsbf

1 uimsbf

1 uimsbf

FIG 4T	FIG 4T-1
	FIG 4T-2
	FIG 4T-3

FIG 4T-2

20/34

```
    } else {  
        tns_present_both;                                1    uimsbf  
        if (tns_present_both) {  
            tns_data_present[0] = 1;  
            tns_data_present[1] = 1;  
        } else {  
            tns_data_present[1];                          1    uimsbf  
            tns_data_present[0] = 1 - tns_data_present[1];  
        }  
    }  
} else {  
    common_tns = 0;  
    tns_data_present[0] = 0;  
    tns_data_present[1] = 0;  
}  
} else {  
    common_window = 0;  
    common_tw = 0;  
}  
}
```

FIG 4T	FIG 4T-1
	FIG 4T-2
	FIG 4T-3

FIG 4T-3

21/34

Syntax of fd_channel_stream()

Syntax	No. of bits	Mnemonic
fd_channel_stream(common_window, common_tw, tns_data_present, noiseFilling, indepFlag)		
{		
global_gain;	8	uimsbf
if (noiseFilling) {		
noise_level;	3	uimsbf
noise_offset;	5	uimsbf
}		
else {		
noise_level = 0;		
}		
if (!common_window) {		
ics_info();		
}		
if (tw_mdct) {		
if (!common_tw) {		
tw_data ();		
}		
}		
scala_factor_data ();		
if (tns_data_present) {		
tns_data ();		
{		
ac_spectral_data(indepFlag);		
fac_data_present;		
if (fac_data_present) {		
fac_length = (window_sequence==EIGHT_SHORT_SEQUENCE) ? ccll/16 : ccll/8;		
fac_data(1, fac_length),		
}		
}		
	1	uimsbf

FIG 4U

22/34

Syntax of lpd_channel_stream()

Syntax	No. of bits	Mnemonic
lpd_channel_stream(indepFlag)		
{		
acelp_core_mode;	3	uimsbf
lpd_mode;	5	uimsbf,
		NOTE 1
bpf_control_info;	1	uimsbf
core_mode_last;	1	uimsbf
fac_data_present;	1	uimsbf
first_lpd_flag = !core_mode_last;		
first_tcx_flag=TRUE;		
k = 0;		
if (first_lpd_flag) { last_lpd_mode = -1; }		NOTE 2
while (k < 4) {		
if (k==0){		
if ((core_mode_last==1) && (fac_data_present==1)) {		
fac_data(0, ccfl/8);		
}		
} else {		
if ((last_lpd_mode==0 && mod[k]>0)		
(last_lpd_mode>0 && mod[k]==0)) {		
fac_data(0, ccfl/8);		
}		
}		
}		

FIG 4V	FIG 4V-1
	FIG 4V-2

FIG 4V-1

23/34

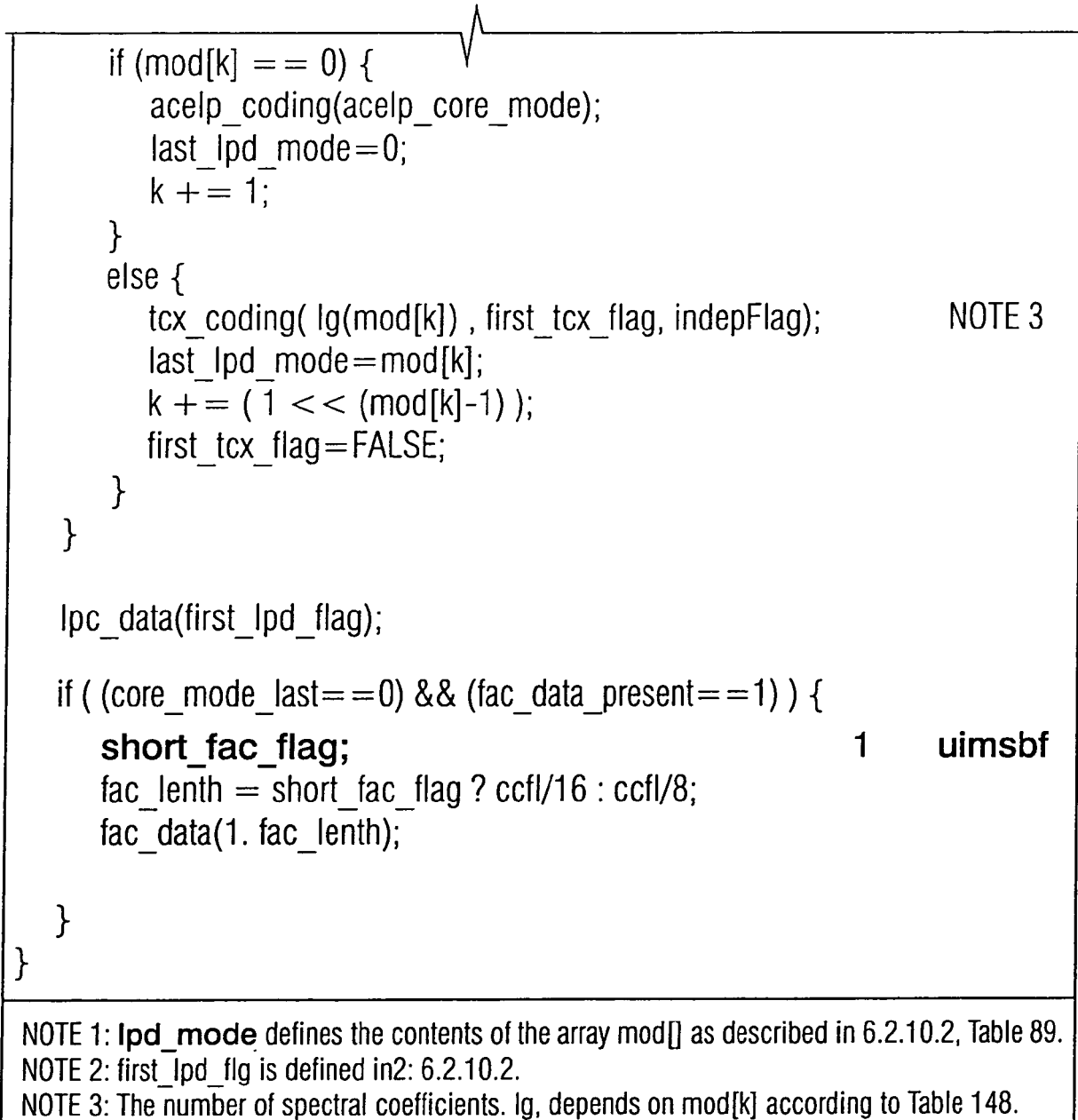


FIG 4V	FIG 4V-1
	FIG 4V-2

FIG 4V-2

24/34

Syntax of fac_data()

Syntax	No. of bits	Mnemonic
<pre> fac_data(useGain, fac_length) { if (useGain) { fac_gain; } for (i=0; i<fac_length/8; i++) { code_book_indices (i, 1, 1); } } </pre>	7	uimsbf
NOTE 1: This value is encoded using a modified unary code, where $q_n=0$ is represented by one "0" bit, and any value q_n greater or equal to 2 is represented by q_n-1 "1" bits followed by one "0" stop bit. Note that $q_n=1$ cannot be signaled, because the codebook Q_7 is not defined.		

FIG 4W

25/34

Syntax of UsacSbrData()

Syntax	No. of bits	Mnemonic
<pre> UsacSbrData(numberSbrChannels, indepFlag) { if (indepFlag) { sbrInfoPresent = 1; sbrHeaderPresent = 1; } else { sbrInfoPresent; if (sbrInfoPresent) { sbrHeaderPresent; } else { sbrHeaderPresent = 0; } } if /sbrInfoPresent) { SbrInfo(); } if (sbrHeaderPresent) { sbrUseDfltHeader; if (sbrUseDfltHeader) { /* copy all SbrDfltHeader() elements dlft_xxx_yyy to bs_xxx_yyy */ } else { SbrHeader(); } } sbr_data(bs_amp_res, numberSbrChannels, indepFlag); } </pre>	1 1 1	uimsbf uimsbf uimsbf

FIG 4X

26/34

Syntax of SbrInfo()

Syntax	No. of bits	Mnemonic
SbrInfo() { bs_amp_res; bs_xover_band; bs_sbr_preprocessing; if (bs_pvc) { bs_pvc_mode; } }	 1 4 1 2	 uimsbf uimsbf uimsbf uimsbf

FIG 4Y

27/34

Syntax of SbrHeader()

Syntax	No. of bits	Mnemonic
SbrHeader() {		
bs_start_freq;	4	uimsbf, NOTE 1
bs_stop_freq;	4	uimsbf, NOTE 1
bs_header_extra_1;	1	uimsbf
bs_header_extra_2;	1	uimsbf
if (bs_header_extra_1) {		NOTE 2
bs_freq_scale;	2	uimsbf
bs_alter_scale;	1	uimsbf
bs_noise_bands;	2	uimsbf
}		
if (bs_header_extra_2) {		NOTE 2
bs_limiter_bands;	2	uimsbf
bs_limiter_gains;	2	uimsbf
bs_interpol_freq;	1	uimsbf
bs_smoothing_mode;	1	uimsbf
}		
}		
NOTE 1: bs_start_freq and bs_stop_freq shall define a frequency band that does not exceed the limits defined in 7.5.5 and ISO/IEC 14496-3:2009, 4.6.18.3.6.		
NOTE 2: If this bit is not set the default values for the underlying data elements shall be used disregarding any previous value.		

FIG 4Z

28/34

Syntax of sbr_data()

Syntax	No. of bits	Mnemonic
sbr_data(bs_amp_res, numberSbrChannels, indepFlag)		
{		
switch (numberSbrChannels) {		
case 1:		
sbr_single_channel_element(bs_amp_res, bs_pvc_mode, indepFlag);		
break;		
case 2:		
sbr_channel_pair_element(bs_amp_res, indepFlag);		
break;		
}		
}		

FIG 4ZA

29/34

Syntax of ssbr_envelope()

Syntax	No. of bits	Mnemonic
<pre> sbr_envelope(ch, bs_coupling, bs_amp_res) { if (bs_coupling) { if (ch) { if (bs_amp_res) { t_huff = t_huffman_env_bal_3_0dB; f_huff = f_huffman_env_bal_3_0dB; } else { t_huff = t_huffman_env_bal_1_5dB; f_huff = f_huffman_env_bal_1_5dB; } } else { if (bs_amp_res) { t_huff = t_huffman_env_3_0dB; f_huff = f_huffman_env_3_0dB; } else { t_huff = t_huffman_env_1_5dB; f_huff = f_huffman_env_1_5dB; } } } else { if (bs_amp_res) { t_huff = t_huffman_env_3_0dB; f_huff = f_huffman_env_3_0dB; } else { t_huff = t_huffman_env_1_5dB; f_huff = f_huffman_env_1_5dB; } } } </pre>		

FIG 4ZB

FIG 4ZB-1

FIG 4ZB-2

FIG 4ZB-1

30/34

<pre> for (env = 0; env < bs_num_env[ch]: env++) { if (bs_df_env[ch][env] == 0) { if (bs_coupling && ch) { if (bs_amp_res) bs_data_env[ch][env][0] = bs_env_start_value_balance; else bs_data_env[ch][env][0] = bs_env_start_value_balance; } else { if (bs_amp_res) bs_data_env[ch][env][0] = bs_env_start_value_level; else bs_data_env[ch][env][0] = bs_env_start_value_level; } for (band = 1:band < num_env_bands[bs_freq_res[ch][env]: band++) bs_data_env[ch][env][band] = sbr_huff_dec(t_huff, bs_codeword); } else { for (band = 0:band < num_env_bands[bs_freq_res[ch][env]: band++) bs_data_env[ch][env][band] = sbr_huff_dec(t_huff, bs_codeword); } if (bs_interTes) { bs_temp_shape[ch][env]; if (bs_temp_shape[ch][env]) { bs_inter_temp_shape_mode[ch][env]; } } } </pre>		
	5	uimsbf
	6	uimsbf
	6	uimsbf
	7	uimsbf
		NOTE 1
	1..18	NOTE 2
		NOTE 1
	1..18	NOTE 2
	1	uimsbf
	2	uimsbf
NOTE 1: num_env_bands[bs_freq_res[ch][env]] is derived from the header according to ISO/IEC 14496-3:2009, 4.6.18.3 and is named n. NOTE 2: sbr_huff_dec() is defined in ISO/IEC 14496-3:2009, 4.A.6.1.		

FIG 4ZB	FIG 4ZB-1
	FIG 4ZB-2

FIG 4ZB-2

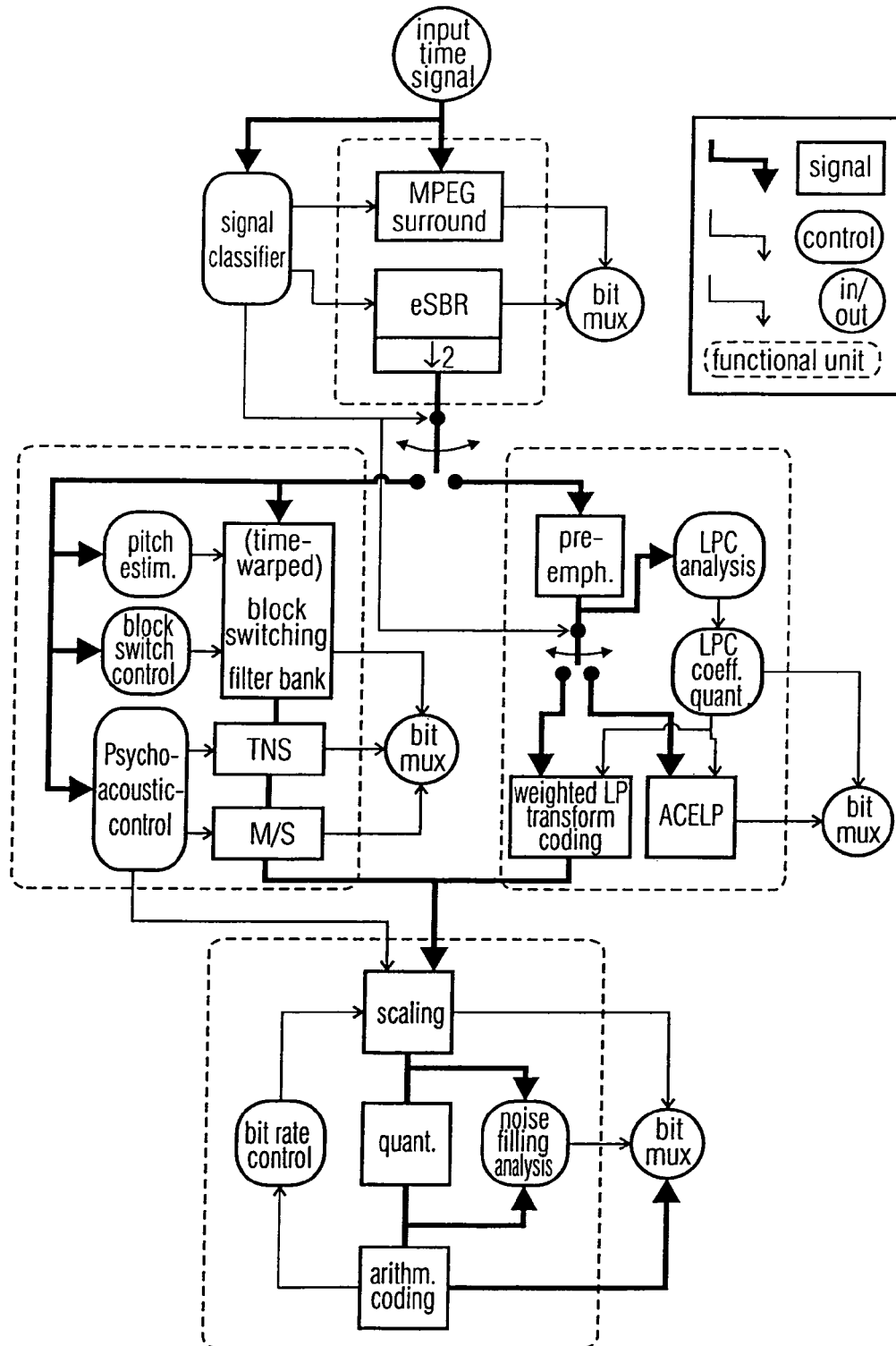
31/34

Syntax of FramingInfo()

Syntax	No. of bits	Mnemonic
<pre> FramingInfo() { if (bsHighRateMode) { bsFramingType; bsNumParamSets; } else { bsFramingType = 0; bsNumParamSets = 1; } numParamSets = bsNumParamSets + 1; nBitsParamSlot = ceil(log2(numSlots)); if (bsFramingType) { for (ps=0; ps<numParamSets; ps++) { bsParamSlot[ps]; } } } </pre>	1 3	uimsbf uimsbf nBitsParamSlot uimsbf

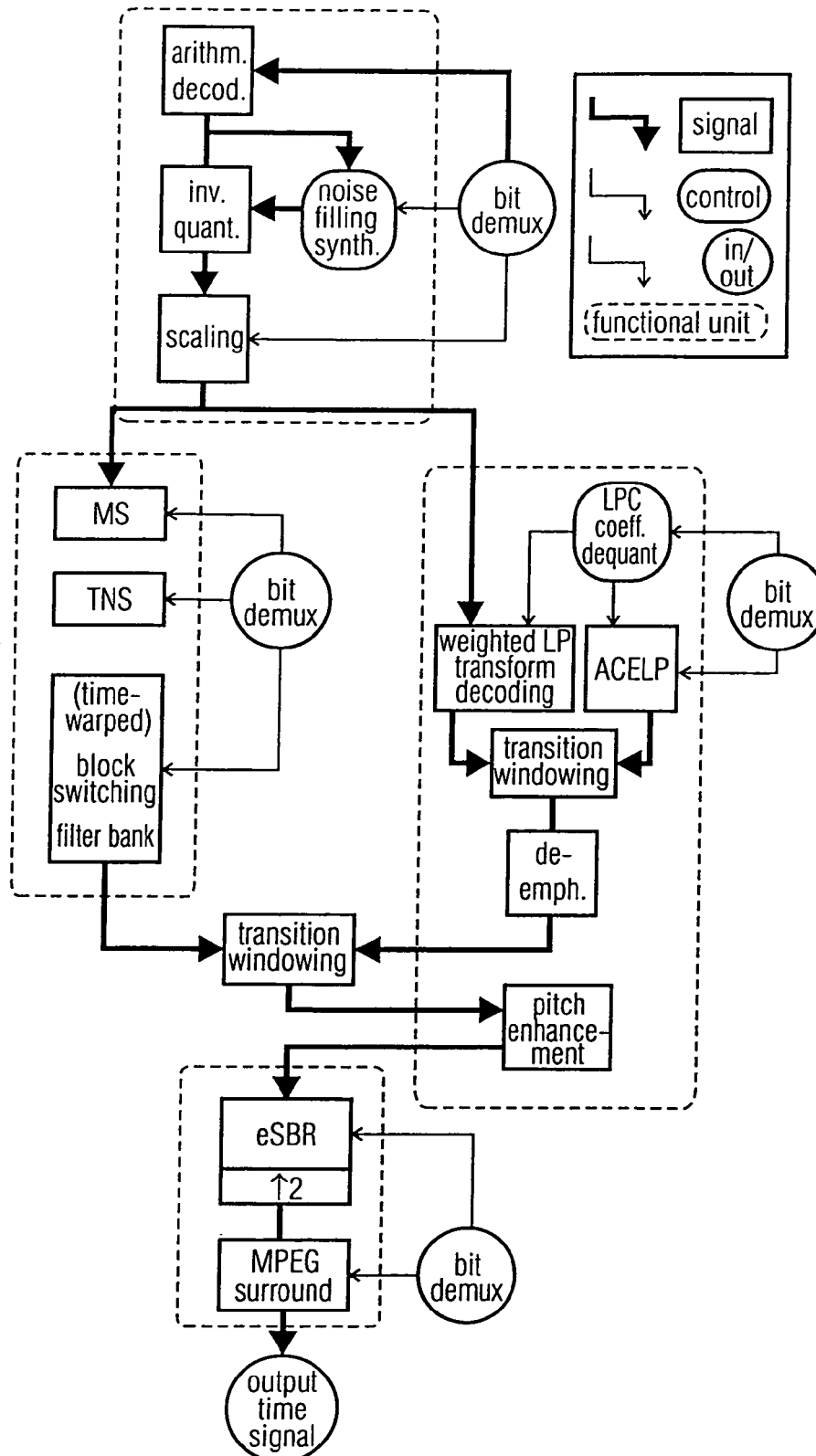
FIG 4ZC

32/34



Block Diagram of the USAC encoder
 FIG 5A
 (PRIOR ART)

33/34



Block Diagram of the USAC decoder
FIG 5B
(PRIOR ART)

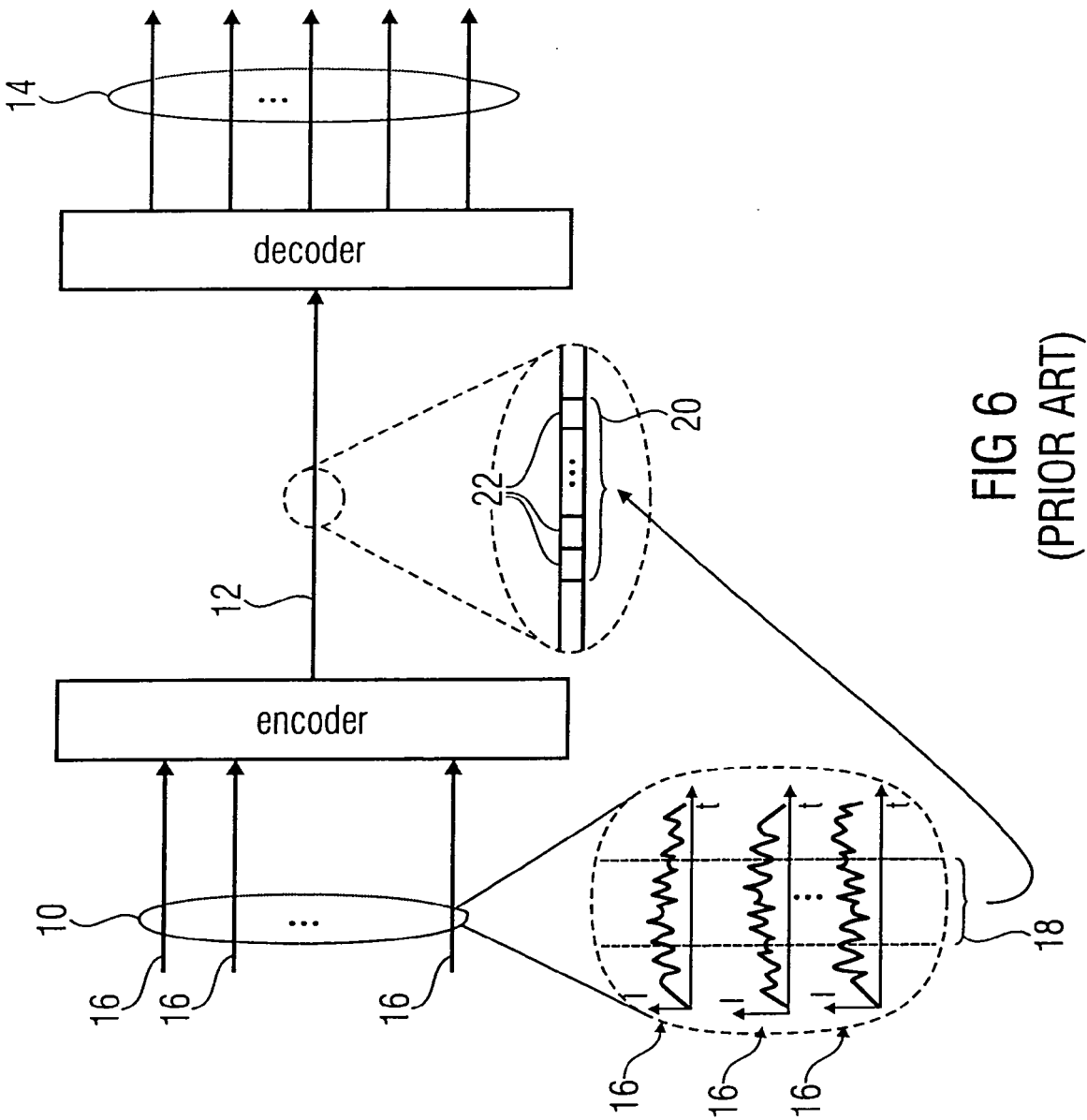


FIG 6
(PRIOR ART)