Subpart 4 of CD 14496-3 is split into several files:

w2203tfs	Syntax, semantics and decoder description
w2203tft	T/F tool descriptions and normative Annex
w2203tfa	This file. Informative annex (Transport streams, Encoder tools)
w2203tvq	Twin-VQ vector quantizer tables

ANNEX A: Encoder

Weighted interleave vector quantization

In the weighted interleave VQ section, the flattened MDCT coefficients vector and the weight vector is divided into subvector at the first stage. Then, the subvectors are applied to weighted vector quantization.

Definition

bits_available_vqNumber of bits available for VQ section.

Initialization

Number of sub-blocks in a frame, N_SF, is set as follows:

```
N_SF = FRAME_SIZE / N_FR
```

Value of N_FR is changed according to bitrate mode and sub-block type (see Tables from TSPN1 to TSPN?). FRAME_SIZE equals to N_FR of long block.

Value of bits_available is calculated as follows:

```
available_vq =
   (int)(FRAME_SIZE*number_of_channel*bitrate/sampling_frequency) -
   bits_for_side_information
```

Number of vector division, N_DIV, and length for each sub-vector length[] are set as follows.

Number of shape sub-vector, `N_DIV`. Length of the codevector, `lengh[idiv]`, are calculated by the following formula:

```
   N_DIV = ((int)((bits_available_vq + MAXBIT*2-1)/(MAXBIT*2)))

   for(idiv=0; idiv<ntt_N_DIV; idiv++){
      length[idiv] = (FRAME_SIZE * n_ch + N_DIV - 1 - idiv) / N_DIV
      bits = (bits_available_vq + N_DIV - 1 - idiv) / N_DIV
      bits0[idiv] = (int)(bits+1) / 2
      bits1[idiv] = (int)bits/2
   }
```

Division of the vectors

At the first stage of encoding process, input vector is divided into subvectors as follows:

```
for (idiv=0; idiv<N_DIV; idiv++){
   for (icv=0; icv<length[idiv]; icv++){
      itmp = idiv + icv * N_DIV
      isf  = itmp % N_SF
      ismp = itmp / N_SF
```

```
      Usub[idiv][icv] = U[isf][ismp]
      wtsub[idiv][icv] = lpcspectrum[ismp]*qenv[isf][ismp]
    }
}
```

where Usub[] and wtsub[] is element of input-coefficients subvector and weight subvector respectively.

Vector quantization

Basic structure

This vector quantizer has a two-channel conjugate structure, where two sets of codebooks are prepared. The search procedure consists of three steps; pre-selection, main selection and index packing.

Pre-selection

At the first step of the vector quantization procedure, candidates for the nearest codevector are selected from each codebook. To select the candidates, weighted distortion measures are calculated.

$$dist[idiv][icb] = \sum_{ismp=0}^{vec\_len[idiv]-1} wt_{sub}{}^{2}(sp\_cv0[icb][ismp] - U_{sub}[idiv][ismp])^{2},$$

$$\text{for } idiv = 0 \text{ to } N\_DIV - 1,\ icb = 0 \text{ to } 2^{bits[idiv]} - 1$$

If the *bits*0[*idiv*] is grater than MAXBIT_SHAPE, the following distortion are also calculated.

$$dist_{negative}[idiv][icb] = \sum_{ismp=0}^{vec\_len[idiv]-1} wt_{sub}{}^{2}(sp\_cv0[icb][ismp] + U_{sub}[idiv][ismp])^{2},$$

$$\text{for } idiv = 0 \text{ to } N\_DIV - 1,\ icb = 0 \text{ to } 2^{bits[idiv]} - 1$$

The N_CAN candidates of dist and $dist_{negative}$ are selected with minimum distortion measure

Main selection

The distortion measures are calculated as below.

$$dist_{cross}[idiv][icb] = \sum_{ismp=0}^{vec\_len[idiv]-1} wt_{sub}[ismp]\left(\begin{array}{l} pol0[idiv] \\ \cdot sp\_cv0[can\_ind0[ican0]][ismp] \\ + pol1[idiv] \\ \cdot sp\_cv1[can\_ind1[ican1]][ismp] \\ \hline 2 \end{array} - U_{sub}[idiv][ismp] \right)^{2}$$ which

$$\text{for } idiv = 0 \text{ to } N\_DIV - 1,\ ican0 = 0 \text{ to } N\_CAN - 1,\ ican1 = 0 \text{ to } N\_CAN - 1$$

$$pol0[idiv] = 1 \quad \text{if } bits0[idiv] = MAXBIT\_SHAPE$$

$$pol0[idiv] = \begin{cases} 1 \\ -1 \end{cases} \quad \text{if } bits0[idiv] > MAXBIT\_SHAPE$$

$$pol1[idiv] = 1 \quad \text{if } bits1[idiv] = MAXBIT\_SHAPE$$

$$pol1[idiv] = \begin{cases} 1 \\ -1 \end{cases} \quad \text{if } bits1[idiv] > MAXBIT\_SHAPE$$

Indexes *can_ind*0[*ican*0] and *can_ind*1[*ican*1] which give the least distortion measure is the quantization indexes *index*0[*idiv*] and *index*1[*idiv*].

Index packing

For each VQ index, the polarity information is added as follows.

If $pol0[idiv] = -1$ then $index0[idiv] = index0[idiv] + 2^{MAXBIT\_SHAPE}$, for $idiv = 0$ to N_DIV

If $pol1[idiv] = -1$ then $index1[idiv] = index1[idiv] + 2^{MAXBIT\_SHAPE}$, for $idiv = 0$ to N_DIV

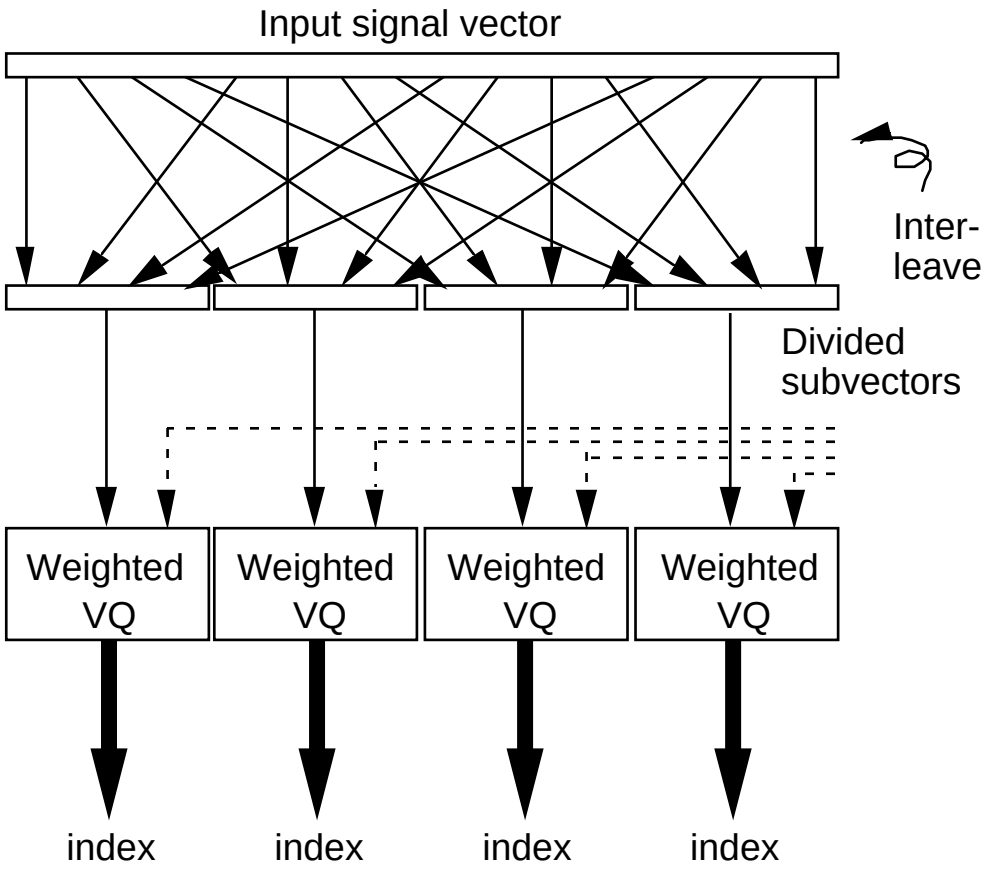


Figure 4. Weighted interleave vector quantization.

Spectrum normaliztion

Definition

Encoding process

Bandwidth control

This process limits the bandwidth of MDCT spectrum.

LPC spectrum coding

At the first stage of the spectrum normalization process, input samples are divided into two paths. In one path they are transformed into frequency domain coefficients by MDCT. In the other path, LPC coefficients are calculated by LPC analysis; then, the LPC modelled spectrum envelope is calculated; and finally, each MDCT coefficient is divided by its corresponding spectrum envelope and first-stage flattened coefficients are produced. The main advantage of this flattening method is that fewer bits are required to normalize the frequency characteristics of input signals as a result of making use of an efficient quantization scheme of LSP (Line Spectral Pair) parameters. In order to avoid square root calculation in the de-flattening procedure at the decoder, LPC coefficients for the square root envelope are quantized; they are derived from the half-value of the LPC cepstrum of the normal LPC coefficients. The procedure for obtaining spectral envelope is listed as follows.

- Autocorrelation function is calculated from the windowed input signal.
- LPC coefficients are calculated by the Levinson-Durbin-Shur method.
- LPC cepstrum coefficients are generated from LPC coefficients.
- LPC cepstrum coefficients are multiplied by 0.5.
- LPC coefficients (corresponding to square root spectrum) are converted from the LPC cepstrum coefficients by solving a normal equation.
- Stability of the obtained coefficients is checked.
- LPC coefficients are converted to LSP parameters.

LSP parameters are quantized by 2-stage split vector quantizer with moving average inter-frame prediction. Inverse LPC spectral envelope values are calculated by the same method in 2.3.1.5.

First-stage flattening

MDCT coefficients $X[isf][j]$ are flattened using LPC spectrum as follows.

$$X_{flat1}[isf][ismp] = X[isf][ismp] / lpc\_spectrum[j],$$
$$\text{for } 0 \le isf < N\_SF, \quad 0 \le ismp < N\_FR$$

Periodic peak components coding

Fundamental frequency estimation

For the low rate coder mode and only for a long frame mode, the periodic peak components of the flattened MDCT coefficients due to the pitch frequency of speech, vocal and some musical instruments.

Vector quantization of the periodic peak components

Extracted pitch components are quantized by the interleaved weighted vector quantization similar to those used for flattened MDCT coefficients.

Subtraction of periodic peak components

Bark-scale envelope coding

Calculation of the Bark-scale envelope

The inputs of the Bark-scale envelope calculation procedure are the MDCT coefficients xflat[] flattened by the LPC spectrum. First the square-rooted power of the input coefficients corresponding to each Bark-scale subband is calculated. The upper frequency boundary of each subband is listed in tables from ? to ?.

$$env\_tmp[isf][ib] = \sqrt{\frac{\sum_{ismp=iblow}^{ibhigh-1} X_{flat1}[ismp]^2}{ibhigh - iblow}},$$
$$\text{for } 0 \le isf < N\_SF,\ 0 \le ib < N\_CRB$$

Next, the average of all square-rooted powers is calculated.

$$env\_avr[isf] = \frac{\sum_{ib=0}^{N\_CRB-1} env\_tmp[isf][ib]}{N\_CRB}, \quad \text{for } 0 \le isf < N\_SF$$

Then, square-rooted powers are normalized by the average to create the MDCT envelope.

$$env[isf][ib] = env\_tmp[isf][ib] / env\_avr[isf],$$
$$\text{for } 0 \le isf < N\_SF,\ 0 \le ib < N\_CRB$$

Weight calculation

The weight is used for the vector quantization of the MDCT envelope.

$$env\_fwt[ib] = \frac{\sum_{ismp=iblow}^{ibhigh-1} (lpc\_spectrum[ismp])}{ibhigh - iblow}$$

Quantization

Interframe prediction

Before quantizing the MDCT envelope, the prediction switch is set. The values of envelope elements are subtracted by their average (is 1).

$$env[is][ib] = env[isf][ib] - 1, \quad \text{for } 0 \le isf < N\_SF,\ 0 \le ib < N\_CRB$$

Correlation between current-frame env[][] and previous-frame env[][] is calculated.

$$correlation = \frac{\sum_{ib=0}^{N\_CRB-1} env_{current}[isf][ib] \cdot env_{previous}[isf][ib]}{\sum_{ib=0}^{N\_CRB-1} (env_{current}[isf][ib])^2}, \quad \text{for } 0 \le isf < N\_SF$$

if N_SF is not equal to 1, the envelope of the previous frame is set by a following equation:

$$env_{previous}[0][ib] = 0,$$
$$env_{previous}[isf][ib] = env_{current}[isf-1][ib], \quad \text{for } 0 \le isf < N\_SF,\ 0 \le ib < N\_CRB$$

If the correlation measure is greater than 0.5, prediction is on, otherwise, prediction is off.

$$\text{if } (correlation[isf] > 0.5) \text{ then } index\_fw\_alf[isf] = 1$$
$$\text{else} \qquad index\_fw\_alf[isf] = 0,$$
$$\text{for } 0 \le isf < N\_SF$$

Vector quantization of the MDCT envelope

At the first step of quantization, the envelope and the weight vector are divided into FW_N_DIV subvectors.

$$denv[isf][ifdiv][icv] = env[isf][FW\_N\_DIV * icv + ifdiv],$$
$$\text{for } 0 \le isf < N\_SF,\ 0 \le ifdiv < N\_CRB,\ 0 \le icv < FW\_CB\_LEN$$
$$dfwt[ifdiv][icv] = env\_fwt[FW\_N\_DIV * icv + ifdiv],$$
$$\text{for } ifdiv = 0 \text{ to } N\_CRB - 1,\ icv = 0 \text{ to } FW\_CB\_LEN - 1$$

Distortion measures $fwdist[itmp]$ are calculated as follows.

$$alfq[isf] = index\_fw\_alf[isf] \cdot FW\_ALF\_STEP$$

$$fwdist[isf][itmp] =$$
$$\sum_{icv=0}^{FW\_CB\_LEN-1} dfwt[ifdiv][icv]^2 \cdot$$
$$(denv[isf][ifdiv][icv] - alfq \cdot p\_cv\_env[isf][icv] - cb\_env[itmp][icv])^2$$
$$\text{for } 0 \le isf < N\_SF,\ 0 \le itmp < FW\_CB\_SIZE,\ 0 \le ifdiv < FW\_N\_DIV$$

The cv_env[][] are elements of the envelope codebook. The p_cv_env[] represents the MDCT envelope of the previous frame. It is calculated by the procedure mensioned in section 2.4.4.3.

Finally, $itmp$ for the minimum distortion measure $fwdist[itmp]$ is the quantization index $index\_env[ifdiv]$.

Local decoding

After the vector quantization, the MDCT envelope is reproduced as follows.

```
for (isf=0; isf<N_SF; isf++){
  alfq[isf] = index_fw_alf[isf] * FW_ALF_STEP
  for (ifdiv=0; ifdiv<FW_N_DIV; ifdiv++){
    for (icv=0; icv<FW_CB_LEN; icv++){
      ienv = FW_N_DIV * icv + ifdiv
      dtmp = cv_env[index_env[isf][ifdiv]][icv]
      qenv_b[isf][ienv] = dtmp + alfq * p_cv_env[isf][icv] + 1
      if (N_SF==1) p_cv_env[isf][icv] = dtmp
      else         p_cv_env[isf-1][icv] = dtmp
    }
  }
}
```

The cv_env[][] are the envelope codebook.

Then, Bark-scale envelope *qenv_b*[*isf*][*ienv*] are projected into linear-scale envelope *qenv*[*isf*][*ismp*].

```
for (isf=0; isf<N_SF; ist++){
  for (ib=0; ib<N_CRB; ib++){
    for (ismp=iblow[ib]; ismp<ibhigh[ib]; ismp++){
      qenv[isf][ismp]=qenv_b[isf][ib];
    }
  }
}
```

Second-stage flattening

$X_{flat2}[][]$ are flattened as follows:

$$X_{flat2}[isf][j] = X_{flat1}[isf][j] / qenv[j] \qquad \text{for } 0 \le isf < N\_SF,\ 0 \le j < N\_FR$$

Gain quantization and amplitude normalization

Before quantizing the flattened MDCT coefficients, their amplitudes are normalized by the gain factor.

The gain factor is calculated as follow:

$$gain[isf] = \frac{\sqrt{\sum_{ismp=0}^{N\_FR-1} X_{flat2}[isf]^2}}{AMP\_NM}, \quad \text{for } 0 \le isf < N\_SF$$

If N_SF == 1, the gain factor is quantized using the following equation:

$$index\_gain[0] = (\text{int}) \frac{\log10(1 + MU \cdot gain[0] / AMP\_MAX)}{\log10(1 + MU)}$$

Then, gain factor is locally decoded.

```
g_temp = index_gain * STEP + STEP / 2
qgain = AMP_MAX * (exp10(g_temp * log10(1.+MU) / AMP_MAX) -1) / MU
qgain /= AMP_NM
```

If N_SF > 1, the global gain and subframe gains are quantized:

$$index\_gain[0] = (\text{int}) \frac{\log10(1 + MU \cdot gain[0] / AMP\_MAX)}{\log10(1 + MU)}$$

Finally, the flattened MDCT coefficients are normalized by the decoded gain factor *qgain*.

$$U[ismp] = X_{flat2}[ismp] / qgain \quad \text{for } ismp = 0 \text{ to } N\_FR - 1$$

Psychoacoustic Model

General

This annex presents the general Psychoacoustic Model for the AAC encoder . The psychoacoustic model calculates the maximum distortion energy which is masked by the signal energy. This energy is called *threshold.* The threshold generation process has three inputs. They are:

1. The shift length for the threshold calculation process is called *iblen.* This *iblen* must remain constant over any particular application of the threshold calculation process. Since it is necessary to calculate thresholds for two different shift lengths, two processes, each running with a fixed shift length, are necessary. For long FFT *iblen* = 1024, for short FFT *iblen* = 128.

2. For each FFT type the newest *iblen* samples of the signal, with the samples delayed (either in the filterbank or psychoacoustic calculation) such that the window of the psychoacoustic calculation is centered in the time-window of the codec time/frequency transform .

3. The sampling rate. There are sets of tables provided for the standard sampling rates. Sampling rate, just as *iblen*, must necessarily remain constant over one implementation of the threshold calculation process.

The output from the psychoacoustic model is :

1. a set of Signal-to-Mask Ratios and thresholds, which are adapted to the encoder as described below,

2. the delayed time domain data (PCM samples) , which are used by the MDCT,

3. the block type for the MDCT (long, start, stop or short type)

4. an estimation of how many bits should be used for encoding in addition to the average available bits.

The delay of the PCM samples is necessary , because if the switch decision algorithm detects an attack, so that *short blocks* have to be used for the actual frame, the *long block* before the *short blocks* has to be 'patched' to a *start block type* in this case.. Before running the model initially, the array used to hold the preceding FFT source data window and the arrays used to hold *r(w)* and *f(w)* should be zeroed to provide a known starting point.

Comments on notation

Throughout this threshold calculation process, three indices for data values are used. These are:

w- indicates that the calculation is indexed by frequency in the FFT spectral line domain. An index of 0 corresponds to the DC term and an index of 1023 corresponds to the spectral line at the Nyquist frequency.

b - indicates that the calculation is indexed in the threshold calculation partition domain. In the case where the calculation includes a convolution or sum in the threshold calculation partition domain, *bb* will be used as the summation variable. Partition numbering starts at 0.

n - indicates that the calculation is indexed in the coder scalefactor band domain. An index of 0 corresponds to the lowest scalefactor band.

The "spreading function"

Several points in the following description refer to the "spreading function". It is calculated by the following method:

if j>=i
tmpx =3,0 (j-i)
else
tmpx = 1.5(j-i)

Where *i* is the Bark value of the signal being spread, *j* is the Bark value of the band being spread into, and *tmpx* is a temporary variable.

$tmpz = 8 * minimum\ ((tmpx\text{-}0{,}5)^2\text{-}2(tmpx\text{-}0{,}5),0)$

Where *tmpz* is a temporary variable, and minimum (a , b) is a function returning the more negative of a or b.

$tmpy = 15.811389 + 7.5(tmpx + 0.474)\text{-}17{,}5(1{,}0+(tmpx + 0.474)^2)^{0{,}5}$

where *tmpy* is another temporary variable.

if (*tmpy*<-100) then {*sprdngf (i , j)*=0} else {*sprdngf (i , j)*=10^((*tmpz* + *tmpy*)/10)}

Steps in threshold calculation

The following are the necessary steps for the calculation of *SMR(n)* and *xmin(n)* used in the coder for long and short FFT.

1. Reconstruct *2 * iblen* samples of the input signal.
iblen new samples are made available at every call to the threshold generator. The threshold generator must store *2 * iblen*

- *iblen* samples, and concatenate those samples to accurately reconstruct *2 * iblen* consecutive samples of the input signal, *s(i)*, where *i* represents the index,
$0 <= i < 2 * iblen$, of the current input stream.

2. Calculate the complex spectrum of the input signal.
First, *s(i)* is windowed by a Hann window, i.e.

*sw(i) =s(i) * (0,5-0,5 * cos((pi *(i+0.5))/ iblen).*

Second, a standard forward FFT of *sw(i)*calculated.

Third, the polar representation of the transform is calculated. *r(w)* and *f(w)* represent the magnitude and phase components of the transformed *sw(i),* respectively.

3. Calculate a predicted *r(w)*and *f(w).*
A predicted magnitude, r_pred(w) and phase, f_pred(w) are calculated from the preceding two threshold calculation blocks *r(w)*and *f(w)*:

*r_pred(w) = 2.0 * r (t-1)-r(t-2)*

*f_pred(w) =2.0 * f(t-1)-f (t-2)*

where *t* represents the current block number, *t-1* indexes the previous block's data, and *t-2* indexes the data from the threshold calculation block before that.

4. Calculate the unpredictability measure *c(w)*:

```
c(w)= (((r(w) * cos(f(w)) - r_pred(w) * cos(f_pred(w)))^2
      + (r(w) * sin(f(w)) - r_pred(w) * sin(f_pred(w)))^2)^0.5 ) /
      (r(w) + abs(r_pred(w))
```

This formula is used for each of the short blocks with the short FFT, for long blocks for the first 6 lines the unpredictability measure is calculated from the long FFT, for the remaining lines the minimum of the unpredictability of all short FFT's is used. If calculation power should be saved, the unpredictability of the upper part of the spectrum can be set to 0.4.

5. Calculate the energy and unpredictability in the threshold calculation partitions.

The energy in each partition, *e(b),* is:

```
do for each partition b:
        e(b) = 0
        do from lower index to upper index w of partition b
                e(b) = e(b) + r(w)^2
        end do
```

(e(b) is used in the M/S-module (see section 2.6.1 of annex B :„Joint Coding“) : e(b) is equal to Xengy with 'X' = [R,L,M,S])
and the weighted unpredictability, *c(b),* is:

```
do for each partition b:
        c(b) = 0
        do from lower index to upper index w of partition b
                c(b) = c(b) + r(w)^2 * c(w)
        end do
end do
```

The threshold calculation partitions provide a resolution of approximately either one FFT line or 1/3 critical band, whichever is wider. At low frequencies, a single line of the FFT will constitute a calculation partition. At high frequencies, many lines will be combined into one calculation partition. A set of partition values is provided for each of the three sampling rates in tables B.2.1.1 to B.2.1.12. These table elements will be used in the threshold calculation process. There are several elements in each table entry:

1. The index of the calculation partition, *b*.

2. The lowest frequency line in the partition, *w_low(b).*

3. The highest frequency line in the partition, *w_high(b)*

4. The median bark value of the partition, *bval(b)*

5. The threshold in quiet qsthr(b)

A largest value of *b*, *bmax*, equal to the largest index, exists for each sampling rate.

6. Convolve the partitioned energy and unpredictability with the spreading function.

```
for each partition b:
        ecb(b) = 0
        do for each partition bb:
                ecb(b) = ecb(b) +e(bb)* sprdngf(bval(bb),bval(b))
        end do
end do


do for each partition b:
        ct(b) = 0
        do for each partition bb:
                ct(b) = ct(b) +c(bb)* sprdngf(bval(bb),bval(b))
        end do
end do
```

Because *ct(b)* is weighted by the signal energy, it must be renormalized to *cb(b)*.

$$cb(b) = ct(b) / ecb(b)$$

Just as this, due to the non-normalized nature of the spreading function, ecb_b should be renormalized and the normalized energy en_b, calculated.

$$en(b) = ecb(b) * rnorm(b)$$

The normalization coefficient, *rnorm(b).* is:

```
do for each partition b
        tmp(b) = 0
        do for each partition bb
                tmp(b) = tmp(b) + sprdngf(bval(bb),bval(b))
        end do
        rnorm(b) = 1/ tmp(b)
end do
```

7. Convert *cb(b)* to *tb(b)* , the tonality index.

$$tb(b) = -0{,}299 - 0{,}43 \log_e (cb(b))$$

Each *tb(b)* is limited to the range of $0<tb(b)<1$.

8. Calculate the required SNR in each partition.

NMT(b) = 6 dB for all *b*. *NMT(b)* is the value for noise masking tone (in dB) for the partition. *TMN(b) = 18 dB* for all *b*. *TMN(b)* is the value for tone masking noise (in dB) .The required signal to noise ratio, *SNR(b)* , is:

*SNR(b) = tb(b) * TMN(b) + (1-tb(b)) * NMT(b)*

9. Calculate the power ratio.

   The power ratio, *bc(b)* , is:

   bc(b) =10^(-SNR(b) /10)

10. Calculation of actual energy threshold, *nb(b)* .

    *nb(b)= en(b) * bc(b)*

    nb(b) is also used in the M/S-module (see chapter 7 „Joint coding“): nb(b) is equal to Xthr with ‘X’=[R,L,M,S]

11. Pre-echo control and threshold in quiet.

    To avoid pre-echoes the pre-echo control is calculated for short and long FFT, the threshold in quiet is also considered here:

    nb_l(b) is the threshold of partition b for the last block , *qsthr(b)* is the threshold in quiet . The dB values of *qsthr(b)* shown in tables B.2.1.1 to B.2.1.12 are relative to the level that a sine wave of + or - ½ lsb has in the FFT used for threshold calculation. The dB values must be converted into the energy domain after considering the FFT normalization actually used.

    *nb(b) = max (qsthr(b), min (nb(b), nb_l(b)*rpelev))*

    rpelev is set to ‘1’ for short blocks and ‘2’ for long blocks

12. The PE is calculated for each block type from the ratio e(b) / nb (b) , where nb(b) is the threshold and e(b) is the energy for each threshold partition :

    ```
    PE = 0
    do for threshold partition b
            PE = PE - ( w_high(b)-w_low(b)) * log10 ( nb(b) / ( e(b) +1) )
    end do
    ```

13. The decision , whether long or short block type is used for encoding is made according to this pseudo code:

    ```
    if PE for long block is greater than switch_pe then
            coding_block_type = short_block_type
    else
            coding_block_type = long_block_type
    end if
    if ( coding_block_type == short_block_type) and ( last_coding_block_type == long_type ) then
            last coding block type = start_type
    else
            last_coding_block_type = short_type
    ```

    The last four lines are necessary since there is no combined stop/start block type in AAC. *switch_pe* is a implementation dependend constant

14. Calculate the signal-to-mask ratios, *SMR(n)* and the codec threshold *xmin(n)*.

    Tables 3.4 to 3.16 (normative part) shows:

    1. The index, *swb*, of the coder partition called scalefactor band.

    2. The offset of mdct line for the scalefactor band *swb_offset_long/short_window*.

    we define the following variable :
    n = swb

w_low(n) = swb_offset_long/short_window(n)
w_high(n) = swb_offset_long/short_window(n+1) - 1

The FFT energy in the scalefactor band, *epart(n)* , is:

do for each scalefactor band ***n***
epart(n) = 0
do for ***w*** *= lower index w_low(n) to n = upper index w_high(n)*
epart(n) = epart (n) + r(w)^2
end do
end do

the threshold for one line of the spectrum is calculated according to:

do for each threshold partition ***b***
thr(all line_indices in this partition ***b*** *)=*
thr (w_low(b),...,w_high(b)) = nb(b) / (w_high(b)+1-w_low(b))
end do

the noise level in the scalefactor band on FFT level , *npart(n)* is calculated as:

do for each scalefactor band ***n***
npart(n) = minimum(thr(w_low(n)),..., thr(w_high(n)))
** (w_high(n)+1-w_low(n))*
end do

Where, in this case, minimum (a,...,z) is a function returning the smallest positive argument of the arguments a...z.

The ratios to be sent to the quantization module, *SMR(n)*, are calculated as:

SMR(n) = epart(n) / npart(n)

For the calculation of coder thresholds xmin(n) the MDCT energy for each scalefactor band is calculated:

do for all scalefactor bands n
codec_e(n) = 0
do for lower index ***i*** *to higher index* ***i*** *of this scalefactor band*
codec_e(n) = codec_e(n) +(mdct_line(i))^2
end do
end do

Then *xmin(n)*, the maximum allowed error energy on MDCT level, can be calculated according to this formula :

*xmin (n) = npart(n) * codec_e(n) / epart (n)*

15. Calculate the bit allocation out of the psychoacoustic entropy (PE).
bit_allocation = pew1 PE + pew2*sqrt(PE);*

for long blocks the constants are defined as:
pew1 = 0.3 , pew2 = 6.0

for short blocks the PE of the eight short blocks is summed up and the constants are :
pew1 = 0.6 , pew2 = 24

then bit_allocation is limited to 0<bit_allocation < 3000 and
more_bits is calculated :

more_bits = bit_allocation - (average_bits - side_info_bits)

Figure B.2.1.1 - block diagram psychoacoustic model

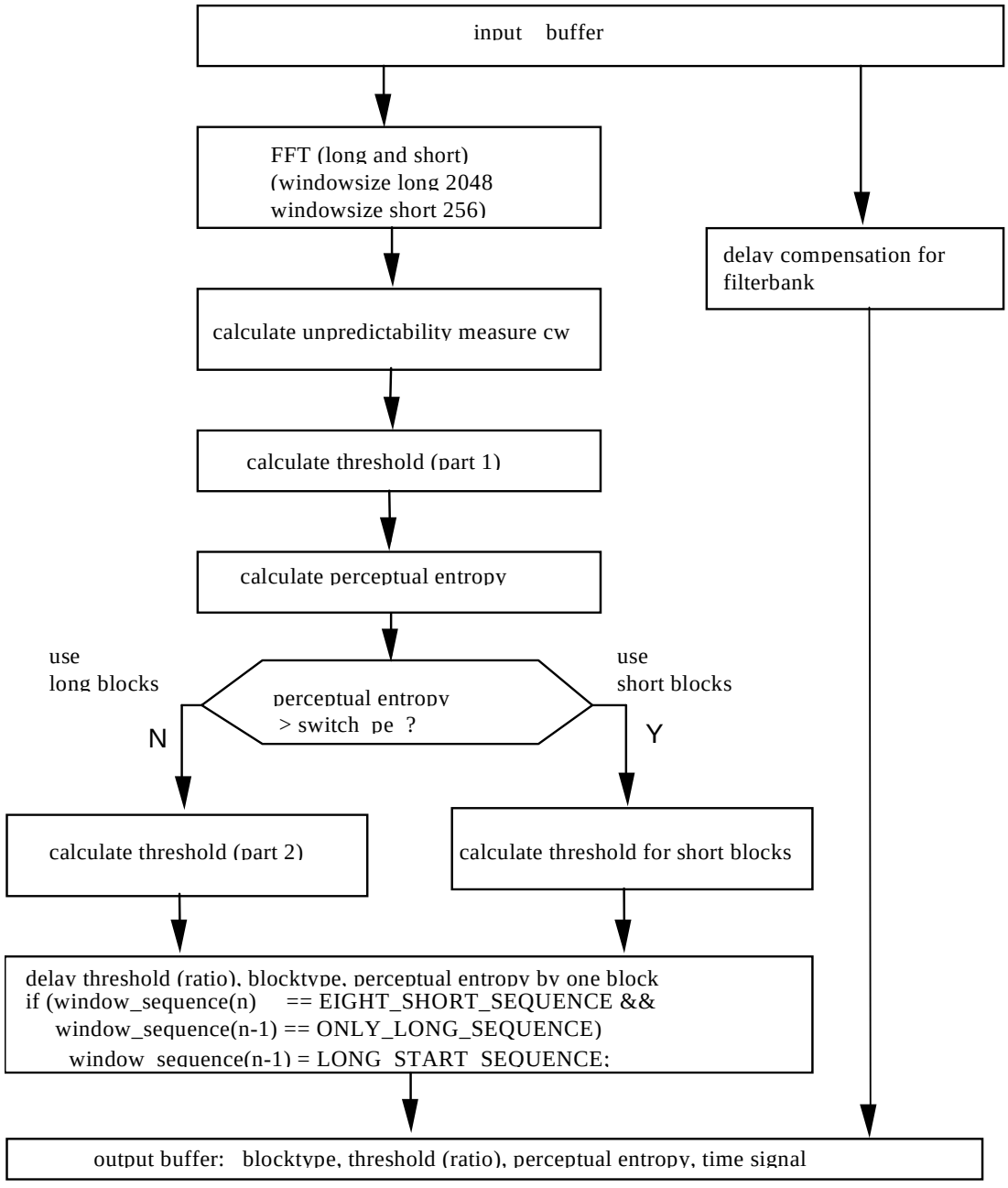


Table B.2.1.1.a -- Psychoacoustic parameters for 8 KHz long FFT

index	w_low	w_high	width	bval	qsthr
0	0	8	9	0,18	46,82
1	9	17	9	0,53	46,82
2	18	26	9	0,89	46,82
3	27	35	9	1,24	41,82
4	36	44	9	1,59	41,82
5	45	53	9	1,94	41,82
6	54	62	9	2,29	38,82
7	63	71	9	2,63	38,82
8	72	80	9	2,98	38,82
9	81	89	9	3,31	33,82
10	90	98	9	3,65	33,82
11	99	108	10	3,99	34,28
12	109	118	10	4,35	32,28
13	119	128	10	4,71	32,28
14	129	138	10	5,05	32,28
15	139	148	10	5,39	32,28
16	149	159	11	5,74	32,69
17	160	170	11	6,10	32,69
18	171	181	11	6,45	32,69
19	182	192	11	6,79	32,69
20	193	204	12	7,13	33,07
21	205	216	12	7,48	33,07
22	217	228	12	7,82	33,07
23	229	241	13	8,17	33,42
24	242	254	13	8,51	33,42
25	255	268	14	8,85	33,74
26	269	282	14	9,20	33,74
27	283	297	15	9,54	34,04
28	298	312	15	9,88	34,04
29	313	328	16	10,22	34,32
30	329	345	17	10,56	34,58
31	346	363	18	10,91	34,83
32	364	381	18	11,25	34,83
33	382	400	19	11,58	35,06
34	401	420	20	11,91	35,29
35	421	441	21	12,24	35,50
36	442	464	23	12,58	35,89
37	465	488	24	12,92	36,08
38	489	514	26	13,26	36,43
39	515	541	27	13,59	36,59
40	542	570	29	13,93	36,90
41	571	601	31	14,26	37,19
42	602	634	33	14,60	37,46
43	635	670	36	14,93	37,84
44	671	708	38	15,27	38,07
45	709	749	41	15,60	38,40
46	750	793	44	15,93	38,71
47	794	841	48	16,26	39,09
48	842	893	52	16,60	39,44
49	894	949	56	16,93	39,76
50	950	1009	60	17,26	40,06
51	1010	1023	14	17,47	33,74

Table B.2.1.1.b -- Psychoacoustic parameters for 8 KHz short FFT

index	w_low	w_high	width	bval	
0	0	1	2	0,32	30,29
1	2	3	2	0,95	30,29
2	4	5	2	1,57	25,29
3	6	7	2	2,19	22,29
4	8	9	2	2,80	22,29
5	10	11	2	3,40	17,29
6	12	13	2	3,99	17,29
7	14	15	2	4,56	15,29
8	16	17	2	5,12	15,29
9	18	19	2	5,66	15,29
10	20	21	2	6,18	15,29
11	22	23	2	6,68	15,29
12	24	25	2	7,16	15,29
13	26	27	2	7,63	15,29
14	28	29	2	8,07	15,29
15	30	31	2	8,50	15,29
16	32	33	2	8,90	15,29
17	34	35	2	9,29	15,29
18	36	37	2	9,67	15,29
19	38	39	2	10,03	15,29
20	40	41	2	10,37	15,29
21	42	44	3	10,77	17,05
22	45	47	3	11,23	17,05
23	48	50	3	11,66	17,05
24	51	53	3	12,06	17,05
25	54	56	3	12,44	17,05
26	57	59	3	12,79	17,05
27	60	63	4	13,18	18,30
28	64	67	4	13,59	18,30
29	68	71	4	13,97	18,30
30	72	75	4	14,32	18,30
31	76	80	5	14,69	19,27
32	81	85	5	15,07	19,27
33	86	90	5	15,42	19,27
34	91	96	6	15,77	20,06
35	97	102	6	16,13	20,06
36	103	109	7	16,49	20,73
37	110	116	7	16,85	20,73
38	117	124	8	17,20	21,31
39	125	127	3	17,44	17,05

Table B.2.1.2.a -- Psychoacoustic parameters for 11,025 KHz long FFT

index	w_low	w_high	width	bval	qsthr
0	0	6	7	0,19	45,73
1	7	13	7	0,57	45,73
2	14	20	7	0,95	45,73
3	21	27	7	1,33	40,73
4	28	34	7	1,71	40,73
5	35	41	7	2,08	37,73
6	42	48	7	2,45	37,73
7	49	55	7	2,82	37,73
8	56	62	7	3,18	32,73
9	63	69	7	3,54	32,73
10	70	76	7	3,89	32,73
11	77	83	7	4,24	30,73
12	84	90	7	4,59	30,73

13	91	97	7	4,92	30,73
14	98	105	8	5,28	31,31
15	106	113	8	5,65	31,31
16	114	121	8	6,01	31,31
17	122	129	8	6,36	31,31
18	130	137	8	6,70	31,31
19	138	146	9	7,06	31,82
20	147	155	9	7,42	31,82
21	156	164	9	7,77	31,82
22	165	173	9	8,11	31,82
23	174	183	10	8,46	32,28
24	184	193	10	8,82	32,28
25	194	203	10	9,16	32,28
26	204	214	11	9,50	32,69
27	215	225	11	9,85	32,69
28	226	237	12	10,19	33,07
29	238	249	12	10,54	33,07
30	250	262	13	10,88	33,42
31	263	275	13	11,22	33,42
32	276	289	14	11,56	33,74
33	290	304	15	11,90	34,04
34	305	320	16	12,24	34,32
35	321	337	17	12,59	34,58
36	338	355	18	12,94	34,83
37	356	374	19	13,28	35,06
38	375	394	20	13,62	35,29
39	395	415	21	13,96	35,50
40	416	438	23	14,29	35,89
41	439	462	24	14,63	36,08
42	463	488	26	14,96	36,43
43	489	516	28	15,29	36,75
44	517	546	30	15,63	37,05
45	547	579	33	15,96	37,46
46	580	614	35	16,30	37,72
47	615	652	38	16,63	38,07
48	653	693	41	16,97	38,40
49	694	737	44	17,30	38,71
50	738	785	48	17,64	39,09
51	786	836	51	17,97	39,35
52	837	891	55	18,30	39,68
53	892	950	59	18,64	39,98
54	951	1014	64	18,97	40,34
55	1015	1023	9	19,16	31,82

Table B.2.1.2.b -- Psychoacoustic parameters for 11,025 KHz short FFT

index	w_low	w_high	width	bval	qsthr
0	0	0	1	0,00	27,28
1	1	1	1	0,44	27,28
2	2	2	1	0,87	27,28
3	3	3	1	1,30	22,28
4	4	4	1	1,73	22,28
5	5	5	1	2,16	19,28
6	6	6	1	2,58	19,28
7	7	7	1	3,00	14,28
8	8	8	1	3,41	14,28
9	9	9	1	3,82	14,28
10	10	10	1	4,22	12,28

11	11	11	1	4,61	12,28
12	12	12	1	4,99	12,28
13	13	13	1	5,37	12,28
14	14	14	1	5,74	12,28
15	15	15	1	6,10	12,28
16	16	16	1	6,45	12,28
17	17	17	1	6,79	12,28
18	18	19	2	7,44	15,29
19	20	21	2	8,05	15,29
20	22	23	2	8,64	15,29
21	24	25	2	9,19	15,29
22	26	27	2	9,70	15,29
23	28	29	2	10,19	15,29
24	30	31	2	10,65	15,29
25	32	33	2	11,08	15,29
26	34	35	2	11,48	15,29
27	36	37	2	11,86	15,29
28	38	39	2	12,22	15,29
29	40	42	3	12,64	17,05
30	43	45	3	13,10	17,05
31	46	48	3	13,53	17,05
32	49	51	3	13,93	17,05
33	52	54	3	14,30	17,05
34	55	58	4	14,69	18,30
35	59	62	4	15,11	18,30
36	63	66	4	15,49	18,30
37	67	70	4	15,84	18,30
38	71	75	5	16,21	19,27
39	76	80	5	16,58	19,27
40	81	85	5	16,92	19,27
41	86	91	6	17,27	20,06
42	92	97	6	17,62	20,06
43	98	104	7	17,97	20,73
44	105	111	7	18,32	20,73
45	112	119	8	18,67	21,31
46	120	127	8	19,02	21,31

Table B.2.1.3.a -- Psychoacoustic parameters for 12 KHz long FFT

index	w_low	w_high	width	bval	qsthr
0	0	5	6	0,18	45,06
1	6	11	6	0,53	45,06
2	12	17	6	0,89	45,06
3	18	23	6	1,24	40,06
4	24	29	6	1,59	40,06
5	30	35	6	1,94	40,06
6	36	41	6	2,29	37,06
7	42	47	6	2,63	37,06
8	48	53	6	2,98	37,06
9	54	59	6	3,31	32,06
10	60	65	6	3,65	32,06
11	66	72	7	4,00	30,73
12	73	79	7	4,38	30,73
13	80	86	7	4,75	30,73
14	87	93	7	5,11	30,73
15	94	100	7	5,47	30,73
16	101	107	7	5,82	30,73
17	108	114	7	6,15	30,73

18	115	122	8	6,51	31,31
19	123	130	8	6,88	31,31
20	131	138	8	7,24	31,31
21	139	146	8	7,58	31,31
22	147	154	8	7,92	31,31
23	155	163	9	8,27	31,82
24	164	172	9	8,62	31,82
25	173	181	9	8,96	31,82
26	182	191	10	9,31	32,28
27	192	201	10	9,66	32,28
28	202	212	11	10,01	32,69
29	213	223	11	10,36	32,69
30	224	235	12	10,71	33,07
31	236	247	12	11,06	33,07
32	248	260	13	11,41	33,42
33	261	273	13	11,75	33,42
34	274	287	14	12,09	33,74
35	288	302	15	12,43	34,04
36	303	318	16	12,77	34,32
37	319	335	17	13,11	34,58
38	336	353	18	13,46	34,83
39	354	372	19	13,80	35,06
40	373	392	20	14,13	35,29
41	393	414	22	14,47	35,70
42	415	437	23	14,81	35,89
43	438	462	25	15,14	36,26
44	463	489	27	15,48	36,59
45	490	518	29	15,81	36,90
46	519	549	31	16,15	37,19
47	550	583	34	16,48	37,59
48	584	619	36	16,82	37,84
49	620	658	39	17,15	38,19
50	659	700	42	17,48	38,51
51	701	745	45	17,81	38,81
52	746	794	49	18,14	39,18
53	795	847	53	18,48	39,52
54	848	904	57	18,81	39,83
55	905	965	61	19,15	40,13
56	966	1023	58	19,47	39,91

Table B.2.1.3.b -- Psychoacoustic parameters for 12 KHz short FFT

index	w_low	w_high	width	bval	qsthr
0	0	0	1	0,00	27,28
1	1	1	1	0,47	27,28
2	2	2	1	0,95	27,28
3	3	3	1	1,42	22,28
4	4	4	1	1,88	22,28
5	5	5	1	2,35	19,28
6	6	6	1	2,81	19,28
7	7	7	1	3,26	14,28
8	8	8	1	3,70	14,28
9	9	9	1	4,14	12,28
10	10	10	1	4,57	12,28
11	11	11	1	4,98	12,28
12	12	12	1	5,39	12,28
13	13	13	1	5,79	12,28
14	14	14	1	6,18	12,28

15	15	15	1	6,56	12,28
16	16	16	1	6,93	12,28
17	17	17	1	7,28	12,28
18	18	18	1	7,63	12,28
19	19	20	2	8,28	15,29
20	21	22	2	8,90	15,29
21	23	24	2	9,48	15,29
22	25	26	2	10,02	15,29
23	27	28	2	10,53	15,29
24	29	30	2	11,00	15,29
25	31	32	2	11,45	15,29
26	33	34	2	11,86	15,29
27	35	36	2	12,25	15,29
28	37	38	2	12,62	15,29
29	39	40	2	12,96	15,29
30	41	43	3	13,36	17,05
31	44	46	3	13,80	17,05
32	47	49	3	14,21	17,05
33	50	52	3	14,59	17,05
34	53	55	3	14,94	17,05
35	56	59	4	15,32	18,30
36	60	63	4	15,71	18,30
37	64	67	4	16,08	18,30
38	68	72	5	16,45	19,27
39	73	77	5	16,83	19,27
40	78	82	5	17,19	19,27
41	83	88	6	17,54	20,06
42	89	94	6	17,90	20,06
43	95	101	7	18,26	20,73
44	102	108	7	18,62	20,73
45	109	116	8	18,97	21,31
46	117	124	8	19,32	21,31
47	125	127	3	19,55	17,05

Table B.2.1.4.a -- Psychoacoustic parameters for 16 KHz long FFT

index	w_low	w_high	width	bval	qsthr
0	0	4	5	0,20	0,20
1	5	9	5	0,59	0,59
2	10	14	5	0,99	0,99
3	15	19	5	1,38	1,38
4	20	24	5	1,77	1,77
5	25	29	5	2,16	2,16
6	30	34	5	2,54	2,54
7	35	39	5	2,92	2,92
8	40	44	5	3,29	3,29
9	45	49	5	3,66	3,66
10	50	54	5	4,03	4,03
11	55	59	5	4,39	4,39
12	60	64	5	4,74	4,74
13	65	69	5	5,09	5,09
14	70	74	5	5,43	5,43
15	75	80	6	5,79	5,79
16	81	86	6	6,18	6,18
17	87	92	6	6,56	6,56
18	93	98	6	6,92	6,92
19	99	104	6	7,28	7,28
20	105	110	6	7,63	7,63

21	111	116	6	7,96	7,96
22	117	123	7	8,31	8,31
23	124	130	7	8,68	8,68
24	131	137	7	9,03	9,03
25	138	144	7	9,37	9,37
26	145	152	8	9,71	9,71
27	153	160	8	10,07	10,07
28	161	168	8	10,41	10,41
29	169	177	9	10,75	10,75
30	178	186	9	11,10	11,10
31	187	196	10	11,45	11,45
32	197	206	10	11,80	11,80
33	207	217	11	12,14	12,14
34	218	228	11	12,48	12,48
35	229	240	12	12,82	12,82
36	241	253	13	13,16	13,16
37	254	267	14	13,51	13,51
38	268	282	15	13,86	13,86
39	283	298	16	14,21	14,21
40	299	315	17	14,56	14,56
41	316	333	18	14,90	14,90
42	334	352	19	15,24	15,24
43	353	373	21	15,58	15,58
44	374	395	22	15,91	15,91
45	396	419	24	16,25	16,25
46	420	445	26	16,58	16,58
47	446	473	28	16,92	16,92
48	474	503	30	17,25	17,25
49	504	536	33	17,59	17,59
50	537	571	35	17,93	17,93
51	572	609	38	18,26	18,26
52	610	650	41	18,60	18,60
53	651	694	44	18,94	18,94
54	695	741	47	19,27	19,27
55	742	791	50	19,60	19,60
56	792	845	54	19,94	19,94
57	846	903	58	20,27	20,27
58	904	965	62	20,61	20,61
59	966	1023	58	20,92	20,92

Table B.2.1.4.b -- Psychoacoustic parameters for 16 KHz short FFT

index	w_low	w_high	width	bval	qsthr
0	0	0	1	0,00	27,28
1	1	1	1	0,63	27,28
2	2	2	1	1,26	22,28
3	3	3	1	1,88	22,28
4	4	4	1	2,50	19,28
5	5	5	1	3,11	14,28
6	6	6	1	3,70	14,28
7	7	7	1	4,28	12,28
8	8	8	1	4,85	12,28
9	9	9	1	5,39	12,28
10	10	10	1	5,92	12,28
11	11	11	1	6,43	12,28
12	12	12	1	6,93	12,28
13	13	13	1	7,40	12,28
14	14	14	1	7,85	12,28

15	15	15	1	8,29	12,28
16	16	16	1	8,70	12,28
17	17	17	1	9,10	12,28
18	18	18	1	9,49	12,28
19	19	19	1	9,85	12,28
20	20	20	1	10,20	12,28
21	21	22	2	10,85	15,29
22	23	24	2	11,44	15,29
23	25	26	2	11,99	15,29
24	27	28	2	12,50	15,29
25	29	30	2	12,96	15,29
26	31	32	2	13,39	15,29
27	33	34	2	13,78	15,29
28	35	36	2	14,15	15,29
29	37	39	3	14,57	17,05
30	40	42	3	15,03	17,05
31	43	45	3	15,45	17,05
32	46	48	3	15,84	17,05
33	49	51	3	16,19	17,05
34	52	55	4	16,57	18,30
35	56	59	4	16,97	18,30
36	60	63	4	17,33	18,30
37	64	68	5	17,71	19,27
38	69	73	5	18,09	19,27
39	74	78	5	18,44	19,27
40	79	84	6	18,80	20,06
41	85	90	6	19,17	20,06
42	91	97	7	19,53	20,73
43	98	104	7	19,89	20,73
44	105	112	8	20,25	24,31
45	113	120	8	20,61	24,31
46	121	127	7	20,92	23,73

Table B.2.1.5.a -- Psychoacoustic parameters for 22,05 KHz long FFT

index	w_low	w_high	width	bval	qsthr
0	0	3	4	0,22	43,30
1	4	7	4	0,65	43,30
2	8	11	4	1,09	38,30
3	12	15	4	1,52	38,30
4	16	19	4	1,95	38,30
5	20	23	4	2,37	35,30
6	24	27	4	2,79	35,30
7	28	31	4	3,21	30,30
8	32	35	4	3,62	30,30
9	36	39	4	4,02	28,30
10	40	43	4	4,41	28,30
11	44	47	4	4,80	28,30
12	48	51	4	5,18	28,30
13	52	55	4	5,55	28,30
14	56	59	4	5,92	28,30
15	60	63	4	6,27	28,30
16	64	67	4	6,62	28,30
17	68	71	4	6,95	28,30
18	72	76	5	7,32	29,27
19	77	81	5	7,71	29,27
20	82	86	5	8,10	29,27
21	87	91	5	8,46	29,27

22	92	96	5	8,82	29,27
23	97	101	5	9,16	29,27
24	102	107	6	9,52	30,06
25	108	113	6	9,89	30,06
26	114	119	6	10,25	30,06
27	120	125	6	10,59	30,06
28	126	132	7	10,95	30,73
29	133	139	7	11,31	30,73
30	140	146	7	11,65	30,73
31	147	154	8	12,00	31,31
32	155	162	8	12,35	31,31
33	163	171	9	12,70	31,82
34	172	180	9	13,05	31,82
35	181	190	10	13,40	32,28
36	191	200	10	13,74	32,28
37	201	211	11	14,07	32,69
38	212	223	12	14,41	33,07
39	224	236	13	14,76	33,42
40	237	250	14	15,11	33,74
41	251	265	15	15,46	34,04
42	266	281	16	15,80	34,32
43	282	298	17	16,14	34,58
44	299	317	19	16,48	35,06
45	318	337	20	16,82	35,29
46	338	359	22	17,16	35,70
47	360	382	23	17,50	35,89
48	383	407	25	17,84	36,26
49	408	434	27	18,17	36,59
50	435	463	29	18,51	36,90
51	464	494	31	18,84	37,19
52	495	527	33	19,17	37,46
53	528	563	36	19,51	37,84
54	564	601	38	19,84	38,07
55	602	642	41	20,17	41,40
56	643	686	44	20,50	41,71
57	687	733	47	20,84	42,00
58	734	784	51	21,17	44,35
59	785	839	55	21,50	44,68
60	840	898	59	21,84	44,98
61	899	962	64	22,17	50,34
62	963	1023	61	22,48	50,13

Table B.2.1.5.b -- Psychoacoustic parameters for 22,05 KHz short FFT

index	w_low	w_high	width	bval	qsthr
0	0	0	1	0,00	27,28
1	1	1	1	0,87	27,28
2	2	2	1	1,73	22,28
3	3	3	1	2,58	19,28
4	4	4	1	3,41	14,28
5	5	5	1	4,22	12,28
6	6	6	1	4,99	12,28
7	7	7	1	5,74	12,28
8	8	8	1	6,45	12,28
9	9	9	1	7,12	12,28
10	10	10	1	7,75	12,28
11	11	11	1	8,36	12,28
12	12	12	1	8,92	12,28

13	13	13	1	9,45	12,28
14	14	14	1	9,96	12,28
15	15	15	1	10,43	12,28
16	16	16	1	10,87	12,28
17	17	17	1	11,29	12,28
18	18	18	1	11,68	12,28
19	19	19	1	12,05	12,28
20	20	21	2	12,71	15,29
21	22	23	2	13,32	15,29
22	24	25	2	13,86	15,29
23	26	27	2	14,35	15,29
24	28	29	2	14,80	15,29
25	30	31	2	15,21	15,29
26	32	33	2	15,58	15,29
27	34	35	2	15,93	15,29
28	36	38	3	16,32	17,05
29	39	41	3	16,75	17,05
30	42	44	3	17,15	17,05
31	45	47	3	17,51	17,05
32	48	51	4	17,89	18,30
33	52	55	4	18,30	18,30
34	56	59	4	18,67	18,30
35	60	63	4	19,02	18,30
36	64	68	5	19,37	19,27
37	69	73	5	19,74	19,27
38	74	78	5	20,09	22,27
39	79	84	6	20,44	23,06
40	85	90	6	20,79	23,06
41	91	97	7	21,15	25,73
42	98	104	7	21,50	25,73
43	105	112	8	21,85	26,31
44	113	120	8	22,20	31,31
45	121	127	7	22,49	30,73

Table B.2.1.6.a -- Psychoacoustic parameters for 24 KHz long FFT

index	w_low	w_high	width	bval	qsthr
0	0	2	3	0,18	42,05
1	3	5	3	0,53	42,05
2	6	8	3	0,89	42,05
3	9	11	3	1,24	37,05
4	12	14	3	1,59	37,05
5	15	17	3	1,94	37,05
6	18	20	3	2,29	34,05
7	21	23	3	2,63	34,05
8	24	26	3	2,98	34,05
9	27	29	3	3,31	29,05
10	30	32	3	3,65	29,05
11	33	36	4	4,03	28,30
12	37	40	4	4,46	28,30
13	41	44	4	4,88	28,30
14	45	48	4	5,29	28,30
15	49	52	4	5,69	28,30
16	53	56	4	6,08	28,30
17	57	60	4	6,46	28,30
18	61	64	4	6,83	28,30
19	65	68	4	7,19	28,30
20	69	72	4	7,54	28,30

21	73	76	4	7,88	28,30
22	77	81	5	8,25	29,27
23	82	86	5	8,64	29,27
24	87	91	5	9,02	29,27
25	92	96	5	9,38	29,27
26	97	101	5	9,73	29,27
27	102	107	6	10,09	30,06
28	108	113	6	10,47	30,06
29	114	119	6	10,83	30,06
30	120	125	6	11,18	30,06
31	126	132	7	11,53	30,73
32	133	139	7	11,89	30,73
33	140	146	7	12,23	30,73
34	147	154	8	12,57	31,31
35	155	162	8	12,92	31,31
36	163	171	9	13,26	31,82
37	172	180	9	13,61	31,82
38	181	190	10	13,95	32,28
39	191	201	11	14,29	32,69
40	202	213	12	14,65	33,07
41	214	225	12	15,00	33,07
42	226	238	13	15,33	33,42
43	239	252	14	15,66	33,74
44	253	267	15	16,00	34,04
45	268	284	17	16,34	34,58
46	285	302	18	16,69	34,83
47	303	321	19	17,02	35,06
48	322	342	21	17,36	35,50
49	343	364	22	17,70	35,70
50	365	388	24	18,03	36,08
51	389	414	26	18,37	36,43
52	415	442	28	18,70	36,75
53	443	472	30	19,04	37,05
54	473	504	32	19,38	37,33
55	505	538	34	19,71	37,59
56	539	575	37	20,04	40,96
57	576	614	39	20,38	41,19
58	615	656	42	20,71	41,51
59	657	701	45	21,04	43,81
60	702	750	49	21,37	44,18
61	751	803	53	21,70	44,52
62	804	860	57	22,04	49,83
63	861	922	62	22,37	50,20
64	923	989	67	22,70	50,54
65	990	1023	34	22,95	47,59

Table B.2.1.6.b -- Psychoacoustic parameters for 24 KHz short FFT

index	w_low	w_high	width	bval	qsthr
0	0	0	1	0,00	27,28
1	1	1	1	0,95	27,28
2	2	2	1	1,88	22,28
3	3	3	1	2,81	19,28
4	4	4	1	3,70	14,28
5	5	5	1	4,57	12,28
6	6	6	1	5,39	12,28
7	7	7	1	6,18	12,28
8	8	8	1	6,93	12,28

9	9	9	1	7,63	12,28
10	10	10	1	8,29	12,28
11	11	11	1	8,91	12,28
12	12	12	1	9,49	12,28
13	13	13	1	10,03	12,28
14	14	14	1	10,53	12,28
15	15	15	1	11,01	12,28
16	16	16	1	11,45	12,28
17	17	17	1	11,87	12,28
18	18	18	1	12,26	12,28
19	19	19	1	12,62	12,28
20	20	21	2	13,28	15,29
21	22	23	2	13,87	15,29
22	24	25	2	14,40	15,29
23	26	27	2	14,88	15,29
24	28	29	2	15,32	15,29
25	30	31	2	15,71	15,29
26	32	33	2	16,08	15,29
27	34	36	3	16,49	17,05
28	37	39	3	16,94	17,05
29	40	42	3	17,35	17,05
30	43	45	3	17,73	17,05
31	46	48	3	18,07	17,05
32	49	52	4	18,44	18,30
33	53	56	4	18,83	18,30
34	57	60	4	19,20	18,30
35	61	65	5	19,57	19,27
36	66	70	5	19,96	19,27
37	71	75	5	20,31	22,27
38	76	81	6	20,67	23,06
39	82	87	6	21,04	25,06
40	88	94	7	21,41	25,73
41	95	101	7	21,77	25,73
42	102	109	8	22,13	31,31
43	110	117	8	22,48	31,31
44	118	126	9	22,82	31,82
45	127	127	1	23,01	32,28

Table B.2.1.7.a -- Psychoacoustic parameters for 32 KHz long FFT

Index	w_low	w_high	width	bval	qsthr
0	0	2	3	0.24	42.05
1	3	5	3	0.71	42.05
2	6	8	3	1.18	37.05
3	9	11	3	1.65	37.05
4	12	14	3	2.12	34.05
5	15	17	3	2.58	34.05
6	18	20	3	3.03	29.05
7	21	23	3	3.48	29.05
8	24	26	3	3.92	29.05
9	27	29	3	4.35	27.05
10	30	32	3	4.77	27.05
11	33	35	3	5.19	27.05
12	36	38	3	5.59	27.05
13	39	41	3	5.99	27.05
14	42	44	3	6.37	27.05
15	45	47	3	6.74	27.05

16	48	50	3	7.10	27.05
17	51	53	3	7.45	27.05
18	54	56	3	7.80	27.05
19	57	60	4	8.18	28.30
20	61	64	4	8.60	28.30
21	65	68	4	9.00	28.30
22	69	72	4	9.39	28.30
23	73	76	4	9.76	28.30
24	77	80	4	10.11	28.30
25	81	84	4	10.45	28.30
26	85	89	5	10.81	29.27
27	90	94	5	11.19	29.27
28	95	99	5	11.55	29.27
29	100	104	5	11.90	29.27
30	105	110	6	12.25	30.06
31	111	116	6	12.62	30.06
32	117	122	6	12.96	30.06
33	123	129	7	13.31	30.73
34	130	136	7	13.66	30.73
35	137	144	8	14.01	31.31
36	145	152	8	14.36	31.31
37	153	161	9	14.71	31.82
38	162	171	10	15.07	32.28
39	172	181	10	15.42	32.28
40	182	192	11	15.76	32.69
41	193	204	12	16.10	33.07
42	205	217	13	16.45	33.42
43	218	231	14	16.80	33.74
44	232	246	15	17.14	34.04
45	247	262	16	17.48	34.32
46	263	279	17	17.82	34.58
47	280	298	19	18.15	35.06
48	299	318	20	18.49	35.29
49	319	340	22	18.84	35.70
50	341	363	23	19.17	35.89
51	364	388	25	19.51	36.26
52	389	415	27	19.85	36.59
53	416	444	29	20.19	39.90
54	445	475	31	20.53	40.19
55	476	508	33	20.87	40.46
56	509	543	35	21.20	42.72
57	544	581	38	21.53	43.07
58	582	622	41	21.86	43.40
59	623	667	45	22.20	48.81
60	668	715	48	22.53	49.09
61	716	768	53	22.86	49.52
62	769	826	58	23.20	59.91
63	827	890	64	23.53	60.34
64	891	961	71	23.86	60.79
65	962	1023	62	24.00	65.89

Table B.2.1.7.b -- Psychoacoustic parameters for 32 KHz short FFT

Index	w_low	w_high	width	bval	qsthr
0	0	0	1	0.00	27.28
1	1	1	1	1.26	22.28
2	2	2	1	2.50	19.28

3	3	3	1	3.70	14.28
4	4	4	1	4.85	12.28
5	5	5	1	5.92	12.28
6	6	6	1	6.93	12.28
7	7	7	1	7.85	12.28
8	8	8	1	8.70	12.28
9	9	9	1	9.49	12.28
10	10	10	1	10.20	12.28
11	11	11	1	10.85	12.28
12	12	12	1	11.45	12.28
13	13	13	1	12.00	12.28
14	14	14	1	12.50	12.28
15	15	15	1	12.96	12.28
16	16	16	1	13.39	12.28
17	17	17	1	13.78	12.28
18	18	18	1	14.15	12.28
19	19	20	2	14.80	15.29
20	21	22	2	15.38	15.29
21	23	24	2	15.89	15.29
22	25	26	2	16.36	15.29
23	27	28	2	16.77	15.29
24	29	30	2	17.15	15.29
25	31	32	2	17.50	15.29
26	33	35	3	17.90	17.05
27	36	38	3	18.34	17.05
28	39	41	3	18.74	17.05
29	42	44	3	19.11	17.05
30	45	48	4	19.50	18.30
31	49	52	4	19.92	18.30
32	53	56	4	20.30	21.30
33	57	60	4	20.65	21.30
34	61	65	5	21.02	24.27
35	66	70	5	21.40	24.27
36	71	75	5	21.75	24.27
37	76	81	6	22.10	30.06
38	82	87	6	22.45	30.06
39	88	94	7	22.80	30.73
40	95	102	8	23.16	41.31
41	103	110	8	23.51	41.31
42	111	119	9	23.85	41.82
43	120	127	8	24.00	60.47

Table B.2.1.8.a -- Psychoacoustic parameters for 44.1 KHz long FFT

Index	w_low	w_high	width	bval	qsthr
0	0	1	2	0.22	40.29
1	2	3	2	0.65	40.29
2	4	5	2	1.09	35.29
3	6	7	2	1.52	35.29
4	8	9	2	1.95	35.29
5	10	11	2	2.37	32.29
6	12	13	2	2.79	32.29
7	14	15	2	3.21	27.29
8	16	17	2	3.62	27.29
9	18	19	2	4.02	25.29
10	20	21	2	4.41	25.29
11	22	23	2	4.80	25.29

12	24	25	2	5.18	25.29
13	26	27	2	5.55	25.29
14	28	29	2	5.92	25.29
15	30	31	2	6.27	25.29
16	32	33	2	6.62	25.29
17	34	35	2	6.95	25.29
18	36	38	3	7.36	27.05
19	39	41	3	7.83	27.05
20	42	44	3	8.28	27.05
21	45	47	3	8.71	27.05
22	48	50	3	9.12	27.05
23	51	53	3	9.52	27.05
24	54	56	3	9.89	27.05
25	57	59	3	10.25	27.05
26	60	62	3	10.59	27.05
27	63	66	4	10.97	28.30
28	67	70	4	11.38	28.30
29	71	74	4	11.77	28.30
30	75	78	4	12.13	28.30
31	79	82	4	12.48	28.30
32	83	87	5	12.84	29.27
33	88	92	5	13.22	29.27
34	93	97	5	13.57	29.27
35	98	103	6	13.93	30.06
36	104	109	6	14.30	30.06
37	110	116	7	14.67	30.73
38	117	123	7	15.03	30.73
39	124	131	8	15.40	31.31
40	132	139	8	15.76	31.31
41	140	148	9	16.11	31.82
42	149	157	9	16.45	31.82
43	158	167	10	16.79	32.28
44	168	178	11	17.13	32.69
45	179	190	12	17.48	33.07
46	191	203	13	17.83	33.42
47	204	217	14	18.18	33.74
48	218	232	15	18.52	34.04
49	233	248	16	18.87	34.32
50	249	265	17	19.21	34.58
51	266	283	18	19.54	34.83
52	284	303	20	19.88	35.29
53	304	324	21	20.22	38.50
54	325	347	23	20.56	38.89
55	348	371	24	20.90	39.08
56	372	397	26	21.24	41.43
57	398	425	28	21.57	41.75
58	426	455	30	21.91	42.05
59	456	488	33	22.24	47.46
60	489	524	36	22.58	47.84
61	525	563	39	22.91	48.19
62	564	606	43	23.25	58.61
63	607	653	47	23.58	59.00
64	654	706	53	23.91	59.52
65	707	765	59	24.00	69.98
66	766	832	67	24.00	70.54
67	833	908	76	24.00	71.08
68	909	996	88	24.00	71.72
69	997	1023	27	24.00	72.09

Table B.2.1.8.b -- Psychoacoustic parameters for 44.1 KHz short FFT

Index	w_low	w_high	width	bval	qsthr
0	0	0	1	0.00	27.28
1	1	1	1	1.73	22.28
2	2	2	1	3.41	14.28
3	3	3	1	4.99	12.28
4	4	4	1	6.45	12.28
5	5	5	1	7.75	12.28
6	6	6	1	8.92	12.28
7	7	7	1	9.96	12.28
8	8	8	1	10.87	12.28
9	9	9	1	11.68	12.28
10	10	10	1	12.39	12.28
11	11	11	1	13.03	12.28
12	12	12	1	13.61	12.28
13	13	13	1	14.12	12.28
14	14	14	1	14.59	12.28
15	15	15	1	15.01	12.28
16	16	16	1	15.40	12.28
17	17	17	1	15.76	12.28
18	18	19	2	16.39	15.29
19	20	21	2	16.95	15.29
20	22	23	2	17.45	15.29
21	24	25	2	17.89	15.29
22	26	27	2	18.30	15.29
23	28	29	2	18.67	15.29
24	30	31	2	19.02	15.29
25	32	34	3	19.41	17.05
26	35	37	3	19.85	17.05
27	38	40	3	20.25	20.05
28	41	43	3	20.62	20.05
29	44	47	4	21.01	23.30
30	48	51	4	21.43	23.30
31	52	55	4	21.81	23.30
32	56	59	4	22.15	28.30
33	60	64	5	22.51	29.27
34	65	69	5	22.87	29.27
35	70	75	6	23.23	40.06
36	76	81	6	23.59	40.06
37	82	88	7	23.93	40.73
38	89	96	8	24.00	51.31
39	97	105	9	24.00	51.82
40	106	115	10	24.00	52.28
41	116	127	12	24.00	53.07

Table B.2.1.9.a -- Psychoacoustic parameters for 48 KHz long FFT

Index	w_low	w_high	width	bval	qsthr
0	0	1	2	0.24	40.29
1	2	3	2	0.71	40.29
2	4	5	2	1.18	35.29
3	6	7	2	1.65	35.29
4	8	9	2	2.12	32.29
5	10	11	2	2.58	32.29

6	12	13	2	3.03	27.29
7	14	15	2	3.48	27.29
8	16	17	2	3.92	27.29
9	18	19	2	4.35	25.29
10	20	21	2	4.77	25.29
11	22	23	2	5.19	25.29
12	24	25	2	5.59	25.29
13	26	27	2	5.99	25.29
14	28	29	2	6.37	25.29
15	30	31	2	6.74	25.29
16	32	33	2	7.10	25.29
17	34	35	2	7.45	25.29
18	36	37	2	7.80	25.29
19	38	40	3	8.20	27.05
20	41	43	3	8.68	27.05
21	44	46	3	9.13	27.05
22	47	49	3	9.55	27.05
23	50	52	3	9.96	27.05
24	53	55	3	10.35	27.05
25	56	58	3	10.71	27.05
26	59	61	3	11.06	27.05
27	62	65	4	11.45	28.30
28	66	69	4	11.86	28.30
29	70	73	4	12.25	28.30
30	74	77	4	12.62	28.30
31	78	81	4	12.96	28.30
32	82	86	5	13.32	29.27
33	87	91	5	13.70	29.27
34	92	96	5	14.05	29.27
35	97	102	6	14.41	30.06
36	103	108	6	14.77	30.06
37	109	115	7	15.13	30.73
38	116	122	7	15.49	30.73
39	123	130	8	15.85	31.31
40	131	138	8	16.20	31.31
41	139	147	9	16.55	31.82
42	148	157	10	16.91	32.28
43	158	167	10	17.25	32.28
44	168	178	11	17.59	32.69
45	179	190	12	17.93	33.07
46	191	203	13	18.28	33.42
47	204	217	14	18.62	33.74
48	218	232	15	18.96	34.04
49	233	248	16	19.30	34.32
50	249	265	17	19.64	34.58
51	266	283	18	19.97	34.83
52	284	303	20	20.31	38.29
53	304	324	21	20.65	38.50
54	325	347	23	20.99	38.89
55	348	371	24	21.33	41.08
56	372	397	26	21.66	41.43
57	398	425	28	21.99	41.75
58	426	456	31	22.32	47.19
59	457	490	34	22.66	47.59
60	491	527	37	23.00	47.96
61	528	567	40	23.33	58.30
62	568	612	45	23.67	58.81
63	613	662	50	24.00	69.27

64	663	718	56	24.00	69.76
65	719	781	63	24.00	70.27
66	782	853	72	24.00	70.85
67	854	937	84	24.00	71.52
68	938	1023	86	24.00	70.20

Table B.2.1.9.b -- Psychoacoustic parameters for 48 KHz short FFT

Index	w_low	w_high	width	bval	qsthr
0	0	0	1	0.00	27.28
1	1	1	1	1.88	22.28
2	2	2	1	3.70	14.28
3	3	3	1	5.39	12.28
4	4	4	1	6.93	12.28
5	5	5	1	8.29	12.28
6	6	6	1	9.49	12.28
7	7	7	1	10.53	12.28
8	8	8	1	11.45	12.28
9	9	9	1	12.26	12.28
10	10	10	1	12.96	12.28
11	11	11	1	13.59	12.28
12	12	12	1	14.15	12.28
13	13	13	1	14.65	12.28
14	14	14	1	15.11	12.28
15	15	15	1	15.52	12.28
16	16	16	1	15.90	12.28
17	17	18	2	16.56	15.29
18	19	20	2	17.15	15.29
19	21	22	2	17.66	15.29
20	23	24	2	18.13	15.29
21	25	26	2	18.54	15.29
22	27	28	2	18.93	15.29
23	29	30	2	19.28	15.29
24	31	33	3	19.69	17.05
25	34	36	3	20.14	20.05
26	37	39	3	20.54	20.05
27	40	42	3	20.92	20.05
28	43	45	3	21.27	22.05
29	46	49	4	21.64	23.30
30	50	53	4	22.03	28.30
31	54	57	4	22.39	28.30
32	58	62	5	22.76	29.27
33	63	67	5	23.13	39.27
34	68	73	6	23.49	40.06
35	74	79	6	23.85	40.06
36	80	86	7	24.00	50.73
37	87	94	8	24.00	51.31
38	95	103	9	24.00	51.82
39	104	113	10	24.00	52.28
40	114	125	12	24.00	53.07
41	126	127	1	24.00	53.07

Table B.2.1.10.a -- Psychoacoustic parameters for 64 KHz long FFT

index	w_low	w_high	width	bval	qsthr
0	0	1	2	0,32	40,29
1	2	3	2	0,95	40,29

2	4	5	2	1,57	35,29
3	6	7	2	2,19	32,29
4	8	9	2	2,80	32,29
5	10	11	2	3,40	27,29
6	12	13	2	3,99	27,29
7	14	15	2	4,56	25,29
8	16	17	2	5,12	25,29
9	18	19	2	5,66	25,29
10	20	21	2	6,18	25,29
11	22	23	2	6,68	25,29
12	24	25	2	7,16	25,29
13	26	27	2	7,63	25,29
14	28	29	2	8,07	25,29
15	30	31	2	8,50	25,29
16	32	33	2	8,90	25,29
17	34	35	2	9,29	25,29
18	36	37	2	9,67	25,29
19	38	39	2	10,03	25,29
20	40	41	2	10,37	25,29
21	42	44	3	10,77	27,05
22	45	47	3	11,23	27,05
23	48	50	3	11,66	27,05
24	51	53	3	12,06	27,05
25	54	56	3	12,44	27,05
26	57	59	3	12,79	27,05
27	60	63	4	13,18	28,30
28	64	67	4	13,59	28,30
29	68	71	4	13,97	28,30
30	72	75	4	14,32	28,30
31	76	80	5	14,69	29,27
32	81	85	5	15,07	29,27
33	86	90	5	15,42	29,27
34	91	96	6	15,77	30,06
35	97	102	6	16,13	30,06
36	103	109	7	16,49	30,73
37	110	116	7	16,85	30,73
38	117	124	8	17,20	31,31
39	125	132	8	17,54	31,31
40	133	141	9	17,88	31,82
41	142	151	10	18,23	32,28
42	152	161	10	18,58	32,28
43	162	172	11	18,91	32,69
44	173	184	12	19,25	33,07
45	185	197	13	19,60	33,42
46	198	211	14	19,94	33,74
47	212	226	15	20,29	37,04
48	227	242	16	20,63	37,32
49	243	259	17	20,97	37,58
50	260	277	18	21,31	39,83
51	278	297	20	21,64	40,29
52	298	318	21	21,98	40,50
53	319	341	23	22,31	45,89
54	342	366	25	22,65	46,26
55	367	394	28	22,98	46,75
56	395	424	30	23,32	57,05
57	425	458	34	23,66	57,59
58	459	495	37	23,99	57,96
59	496	537	42	24,00	68,51

60	538	584	47	24,00	69,00
61	585	638	54	24,00	69,60
62	639	701	63	24,00	70,27
63	702	774	73	24,00	70,91
64	775	861	87	24,00	71,67
65	862	966	105	24,00	72,49
66	967	1023	57	24,00	69,83

Table B.2.1.10.b -- Psychoacoustic parameters for 64 KHz short FFT

index	w_low	w_high	width	bval	qsthr
0	0	0	1	0,00	27,28
1	1	1	1	2,50	19,28
2	2	2	1	4,85	12,28
3	3	3	1	6,93	12,28
4	4	4	1	8,70	12,28
5	5	5	1	10,20	12,28
6	6	6	1	11,45	12,28
7	7	7	1	12,50	12,28
8	8	8	1	13,39	12,28
9	9	9	1	14,15	12,28
10	10	10	1	14,81	12,28
11	11	11	1	15,39	12,28
12	12	12	1	15,90	12,28
13	13	13	1	16,36	12,28
14	14	14	1	16,78	12,28
15	15	15	1	17,16	12,28
16	16	17	2	17,82	15,29
17	18	19	2	18,40	15,29
18	20	21	2	18,92	15,29
19	22	23	2	19,39	15,29
20	24	25	2	19,82	15,29
21	26	27	2	20,21	18,29
22	28	29	2	20,57	18,29
23	30	32	3	20,98	20,05
24	33	35	3	21,43	22,05
25	36	38	3	21,84	22,05
26	39	41	3	22,22	27,05
27	42	45	4	22,61	28,30
28	46	49	4	23,02	38,30
29	50	53	4	23,39	38,30
30	54	58	5	23,75	39,27
31	59	63	5	24,00	49,27
32	64	69	6	24,00	50,06
33	70	76	7	24,00	50,73
34	77	84	8	24,00	51,31
35	85	93	9	24,00	51,82
36	94	104	11	24,00	52,69
37	105	117	13	24,00	53,42
38	118	127	10	24,00	52,28

Table B.2.1.11.a -- Psychoacoustic parameters for 88,2 KHz long FFT

index	w_low	w_high	width	bval	qsthr
0	0	0	1	0,00	37,28
1	1	1	1	0,44	37,28
2	2	2	1	0,87	37,28
3	3	3	1	1,30	32,28

4	4	4	1	1,73	32,28
5	5	5	1	2,16	29,28
6	6	6	1	2,58	29,28
7	7	7	1	3,00	24,28
8	8	8	1	3,41	24,28
9	9	9	1	3,82	24,28
10	10	10	1	4,22	22,28
11	11	11	1	4,61	22,28
12	12	12	1	4,99	22,28
13	13	13	1	5,37	22,28
14	14	14	1	5,74	22,28
15	15	15	1	6,10	22,28
16	16	16	1	6,45	22,28
17	17	17	1	6,79	22,28
18	18	19	2	7,44	25,29
19	20	21	2	8,05	25,29
20	22	23	2	8,64	25,29
21	24	25	2	9,19	25,29
22	26	27	2	9,70	25,29
23	28	29	2	10,19	25,29
24	30	31	2	10,65	25,29
25	32	33	2	11,08	25,29
26	34	35	2	11,48	25,29
27	36	37	2	11,86	25,29
28	38	39	2	12,22	25,29
29	40	42	3	12,64	27,05
30	43	45	3	13,10	27,05
31	46	48	3	13,53	27,05
32	49	51	3	13,93	27,05
33	52	54	3	14,30	27,05
34	55	58	4	14,69	28,30
35	59	62	4	15,11	28,30
36	63	66	4	15,49	28,30
37	67	70	4	15,84	28,30
38	71	75	5	16,21	29,27
39	76	80	5	16,58	29,27
40	81	85	5	16,92	29,27
41	86	91	6	17,27	30,06
42	92	97	6	17,62	30,06
43	98	104	7	17,97	30,73
44	105	111	7	18,32	30,73
45	112	119	8	18,67	31,31
46	120	127	8	19,02	31,31
47	128	136	9	19,35	31,82
48	137	146	10	19,71	32,28
49	147	156	10	20,05	35,28
50	157	167	11	20,39	35,69
51	168	179	12	20,73	36,07
52	180	192	13	21,08	38,42
53	193	206	14	21,43	38,74
54	207	221	15	21,77	39,04
55	222	237	16	22,11	44,32
56	238	255	18	22,45	44,83
57	256	274	19	22,80	45,06
58	275	295	21	23,13	55,50
59	296	318	23	23,47	55,89
60	319	344	26	23,81	56,43
61	345	373	29	24,00	66,90

62	374	405	32	24,00	67,33
63	406	442	37	24,00	67,96
64	443	484	42	24,00	68,51
65	485	533	49	24,00	69,18
66	534	591	58	24,00	69,91
67	592	660	69	24,00	70,66
68	661	745	85	24,00	71,57
69	746	851	106	24,00	72,53
70	852	988	137	24,00	73,64
71	989	1023	35	24,00	67,72

Table B.2.1.11.b -- Psychoacoustic parameters for 88,2 KHz short FFT

index	w_low	w_high	width	bval	qsthr
0	0	0	1	0,00	27,28
1	1	1	1	3,41	14,28
2	2	2	1	6,45	12,28
3	3	3	1	8,92	12,28
4	4	4	1	10,87	12,28
5	5	5	1	12,39	12,28
6	6	6	1	13,61	12,28
7	7	7	1	14,59	12,28
8	8	8	1	15,40	12,28
9	9	9	1	16,09	12,28
10	10	10	1	16,69	12,28
11	11	11	1	17,21	12,28
12	12	12	1	17,68	12,28
13	13	13	1	18,11	12,28
14	14	14	1	18,49	12,28
15	15	15	1	18,85	12,28
16	16	17	2	19,48	15,29
17	18	19	2	20,05	18,29
18	20	21	2	20,55	18,29
19	22	23	2	21,01	20,29
20	24	25	2	21,43	20,29
21	26	27	2	21,81	20,29
22	28	29	2	22,15	25,29
23	30	32	3	22,55	27,05
24	33	35	3	22,98	27,05
25	36	38	3	23,36	37,05
26	39	42	4	23,75	38,30
27	43	46	4	24,00	48,30
28	47	51	5	24,00	49,27
29	52	56	5	24,00	49,27
30	57	62	6	24,00	50,06
31	63	69	7	24,00	50,73
32	70	77	8	24,00	51,31
33	78	87	10	24,00	52,28
34	88	99	12	24,00	53,07
35	100	115	16	24,00	54,32
36	116	127	12	24,00	53,07

Table B.2.1.12.a -- Psychoacoustic parameters for 96 KHz long FFT

index	w_low	w_high	width	bval	qsthr
0	0	0	1	0,00	37,28
1	1	1	1	0,47	37,28
2	2	2	1	0,95	37,28

3	3	3	1	1,42	32,28
4	4	4	1	1,88	32,28
5	5	5	1	2,35	29,28
6	6	6	1	2,81	29,28
7	7	7	1	3,26	24,28
8	8	8	1	3,70	24,28
9	9	9	1	4,14	22,28
10	10	10	1	4,57	22,28
11	11	11	1	4,98	22,28
12	12	12	1	5,39	22,28
13	13	13	1	5,79	22,28
14	14	14	1	6,18	22,28
15	15	15	1	6,56	22,28
16	16	16	1	6,93	22,28
17	17	17	1	7,28	22,28
18	18	18	1	7,63	22,28
19	19	20	2	8,28	25,29
20	21	22	2	8,90	25,29
21	23	24	2	9,48	25,29
22	25	26	2	10,02	25,29
23	27	28	2	10,53	25,29
24	29	30	2	11,00	25,29
25	31	32	2	11,45	25,29
26	33	34	2	11,86	25,29
27	35	36	2	12,25	25,29
28	37	38	2	12,62	25,29
29	39	40	2	12,96	25,29
30	41	43	3	13,36	27,05
31	44	46	3	13,80	27,05
32	47	49	3	14,21	27,05
33	50	52	3	14,59	27,05
34	53	55	3	14,94	27,05
35	56	59	4	15,32	28,30
36	60	63	4	15,71	28,30
37	64	67	4	16,08	28,30
38	68	72	5	16,45	29,27
39	73	77	5	16,83	29,27
40	78	82	5	17,19	29,27
41	83	88	6	17,54	30,06
42	89	94	6	17,90	30,06
43	95	101	7	18,26	30,73
44	102	108	7	18,62	30,73
45	109	116	8	18,97	31,31
46	117	124	8	19,32	31,31
47	125	133	9	19,67	31,82
48	134	143	10	20,03	35,28
49	144	153	10	20,38	35,28
50	154	164	11	20,72	35,69
51	165	176	12	21,07	38,07
52	177	189	13	21,42	38,42
53	190	203	14	21,77	38,74
54	204	218	15	22,12	44,04
55	219	234	16	22,46	44,32
56	235	252	18	22,80	44,83
57	253	271	19	23,14	55,06
58	272	292	21	23,47	55,50
59	293	316	24	23,81	56,08
60	317	342	26	24,00	66,43

61	343	372	30	24,00	67,05
62	373	406	34	24,00	67,59
63	407	445	39	24,00	68,19
64	446	490	45	24,00	68,81
65	491	543	53	24,00	69,52
66	544	607	64	24,00	70,34
67	608	685	78	24,00	71,20
68	686	783	98	24,00	72,19
69	784	910	127	24,00	73,31
70	911	1023	113	24,00	72,81

Table B.2.1.12.b -- Psychoacoustic parameters for 96 KHz short FFT

index	w_low	w_high	width	bval	qsthr
0	0	0	1	0,00	27,28
1	1	1	1	3,70	14,28
2	2	2	1	6,93	12,28
3	3	3	1	9,49	12,28
4	4	4	1	11,45	12,28
5	5	5	1	12,96	12,28
6	6	6	1	14,15	12,28
7	7	7	1	15,11	12,28
8	8	8	1	15,90	12,28
9	9	9	1	16,57	12,28
10	10	10	1	17,16	12,28
11	11	11	1	17,67	12,28
12	12	12	1	18,13	12,28
13	13	13	1	18,55	12,28
14	14	14	1	18,93	12,28
15	15	16	2	19,60	15,29
16	17	18	2	20,20	18,29
17	19	20	2	20,73	18,29
18	21	22	2	21,21	20,29
19	23	24	2	21,64	20,29
20	25	26	2	22,03	25,29
21	27	28	2	22,39	25,29
22	29	31	3	22,79	27,05
23	32	34	3	23,23	37,05
24	35	37	3	23,62	37,05
25	38	41	4	24,00	48,30
26	42	45	4	24,00	48,30
27	46	50	5	24,00	49,27
28	51	55	5	24,00	49,27
29	56	61	6	24,00	50,06
30	62	68	7	24,00	50,73
31	69	77	9	24,00	51,82
32	78	88	11	24,00	52,69
33	89	102	14	24,00	53,74
34	103	120	18	24,00	54,83
35	121	127	7	24,00	50,73

Gain Control

Encoding process

The gain control tool consists of a PQF (Polyphase Quadrature Filter), gain detectors and gain modifiers. This tool receives the input time-domain signals and **window_sequence**, and then outputs gain_control_data and a gain controlled signal whose length is equal to the length of the MDCT window. The block diagram for the gain control tool is shown in Figure B.2.2.11.

Due to the characteristics of the PQF filterbank, the order of the MDCT coefficients in each even PQF band needs to be reversed. This is done by reversing the spectral order of the MDCT coefficients, i.e. exchanging the higher frequency MDCT coefficients with the lower frequency MDCT coefficients.

If the gain control tool is used, the configuration of the filterbank tool is changed as follows. In the case of an EIGHT_SHORT_SEQUENCE window_sequence, the number of coefficients for the MDCT is 32 instead of 128 and eight MDCTs are carried out. In the case of other window_sequence values, the number of coefficients for the MDCT is 256 instead of 1024 and one MDCT is carried out. In all cases, the filter bank tool receives a total of 2048 gain controlled signal values per frame, because the input samples have been overlapped.

PQF

The input signal is divided by a PQF into four equal width frequency bands. The coefficients of each band PQF are given as follows.

$$h_i(n) = \frac{1}{4}\cos\left(\frac{(2i+1)(2n+5)\pi}{16}\right)Q(n), \quad 0 \le n \le 95, 0 \le i \le 3$$

where

$$Q(n) = Q(95-n), \quad 48 \le n \le 95$$

and the values of *Q(n)* are the same values as those of the decoder.

Gain detector

The gain detectors produce gain control data which satisfies the bitstream syntax. This information consists of the number of gain changes, the index of gain change positions and the index of gain change level. Note that the output gain control data applies to the previous input time signal. This means that the gain detector has a one frame delay.

The detection of the gain change point is done in the second half of the MDCT window region and in the non-overlapped region (of LONG_START_SEQUENCE and LONG_STOP_SEQUENCE). Thus the number of regions are one for ONLY_LONG_SEQUENCE, two for LONG_START_SEQUENCE and LONG_STOP_SEQUNCE, and eight for EIGHT_SHORT_SEQUENCE.

The samples in each region are divided into subregions, each having eight-tuple samples. Then one value (e.g. peak value of samples) is selected in these subregions. The ratios between the values of subregions and the value of the last subregion are calculated. If the ratio is greater or less than the value of 2 ^ n where n is an integer between -4 to 11, those subregions can be detected as the gain change points of the signals. The subregion number which is detected as the gain change point is set to be the position data. The exponent of the ratio is set to be the gain data. The time resolution of the gain control is approximately 0.7 ms at 48 kHz sampling rate.

Gain modifier

The gain modifier for each PQF band controls the gain of each signal band. The complementary gain control process in the decoder decreases the pre-echo and reconstructs the original signal. A window function for gain control, the Gain Modification Function (GMF), which is defined in the decoding process, is derived from the gains and the gain-changed positions. The gain controlled signals are derived by applying the GMF to the corresponding band signals.

Diagrams

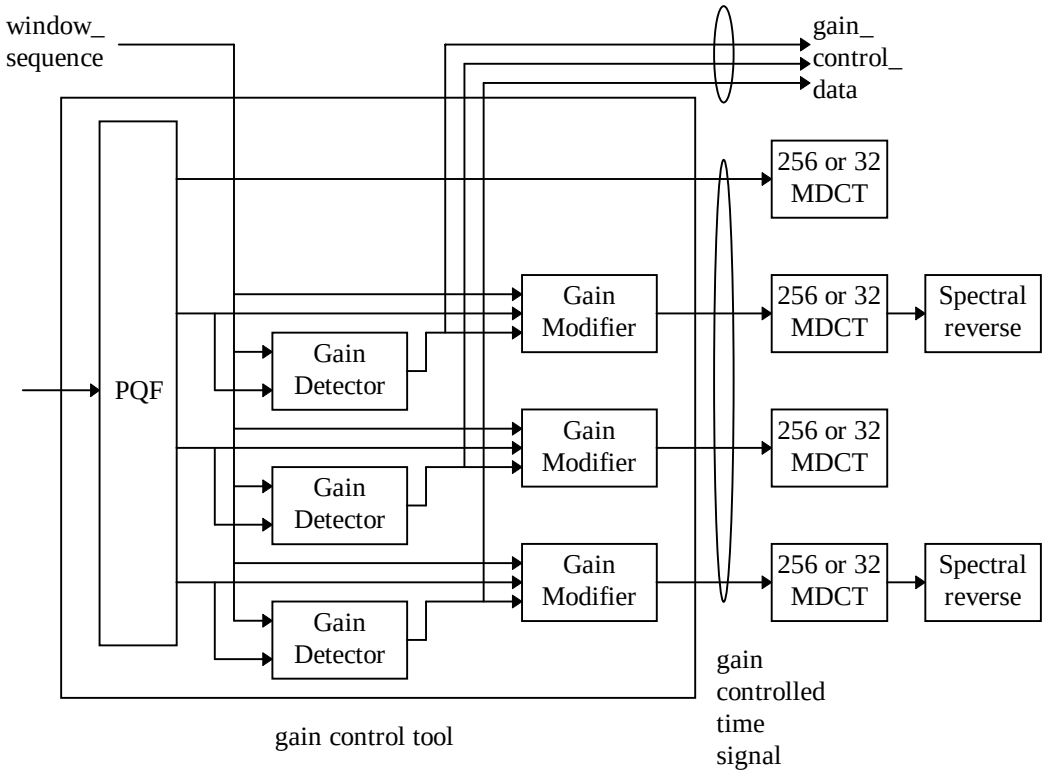


Figure B.2.2.1 -- Block diagram of gain control tool for encoder

Filterbank and Block Switching

A fundamental component in the audio coding process is the conversion of the time domain signals into a time-frequency representation. This conversion is done by a forward modified discrete cosine transform (MDCT).

Encoding process

In the encoder the filterbank takes the appropriate block of time samples, modulates them by an appropriate window function, and performs the MDCT. Each block of input samples is overlapped by 50% with the immediately preceding block and the following block. The transform input block length N can be set to either 2048 or 256 samples. Since the window function has a significant effect on the filterbank frequency response, the filterbank has been designed to allow a change in window shape to best adapt to input signal conditions. The shape of the window is varied simultaneously in the encoder and decoder to allow the filterbank to efficiently separate spectral components of the input for a wider variety of input signals.

Windowing and block switching

The adaptation of the time-frequency resolution of the filterbank to the characteristics of the input signal is done by shifting between transforms whose input lengths are either 2048 or 256 samples. By enabling the block switching tool, the following transitions are meaningful:

from ONLY_LONG_SEQUENCE to	{ONLY_LONG_SEQUENCE LONG_START_SEQUENCE
from LONG_START_SEQUENCE to	{EIGHT_SHORT_SEQUENCE LONG_STOP_SEQUENCE
from LONG_STOP_SEQUENCE to	{ONLY_LONG_SEQUENCE LONG_START_SEQUENCE
from EIGHT_SHORT_SEQUENCE to	{EIGHT_SHORT_SEQUENCE LONG_STOP_SEQUENCE

Window shape decisions are made by the encoder on a frame-by-frame-basis. The window selected is applicable to the second half of the window function only, since the first half is constrained to use the appropriate window shape from the preceding frame. Figure B.2.3.1 shows the sequence of blocks for the transition (D-E-F) to and from a frame employing the sine function window. The window shape selector generally produces window shape run-lengths greater than that shown in the figure.

The 2048 time-domain values $x'_{i,n}$ to be windowed are the last 1024 values of the previous window_sequence concatenated with 1024 values of the current block. The formula below shows this fact:

$$x'_{i,n} = \begin{cases} x_{(i-1),(n+1024)}, & \text{for } 0 \le n < 1024 \\ x_{i,n}, & \text{for } 1024 \le n < 2048 \end{cases}$$

Where i is the block index and n is the sample index within a block. Once the window shape is selected, the **window_shape** syntax element is initialized. Together with the chosen **window_sequence** all information needed for windowing exist.

With the window halves described below all **window_sequences** can be assembled.

For **window_shape** == 1, the window coefficients are given by the Kaiser - Bessel derived (KBD) window as follows:

$$W_{KBD\_LEFT,N}(n) = \sqrt{\frac{\sum_{p=0}^{n}\left[W'(n,\alpha)\right]}{\sum_{p=0}^{N/2}\left[W'(p,\alpha)\right]}} \qquad \text{for} \quad 0 \le n < \frac{N}{2}$$

$$W_{KBD\_RIGHT,N}(n) = \sqrt{\frac{\sum_{p=0}^{N-n}\left[W'(n,\alpha)\right]}{\sum_{p=0}^{N/2}\left[W'(p,\alpha)\right]}} \qquad \text{for} \quad \frac{N}{2} \le n < N$$

where:

W', Kaiser - Bessel kernel window function, see also [5], is defined as follows:

$$W'(n) = \frac{I_0\left[\pi\alpha\sqrt{1.0-\left(\frac{n-N/4}{N/4}\right)^2}\right]}{I_0\left[\pi\alpha\right]} \qquad \text{for } 0 \le n \le \frac{N}{2}$$

$$I_0\left[x\right] = \sum_{k=0}^{\infty}\left[\frac{\left(\frac{x}{2}\right)^k}{k!}\right]^2$$

α = kernel window alpha factor, $\alpha = \begin{cases} 4 \text{ for } N = 2048 \\ 6 \text{ for } N = 256 \end{cases}$

Otherwise, for **window_shape** == 0, a sine window is employed as follows:

$$W_{SIN\_LEFT,N}(n) = \sin(\frac{\pi}{N}(n+\frac{1}{2})) \qquad \text{for} \quad 0 \le n < \frac{N}{2}$$

$$W_{SIN\_RIGHT,N}(n) = \sin(\frac{\pi}{N}(n+\frac{1}{2})) \qquad \text{for} \quad \frac{N}{2} \le n < N$$

The window length N can be 2048 or 256 for the KBD and the sine window. How to obtain the possible window sequences is explained in the parts a)-d) of this clause. All four window_sequences explained below have a total length of 2048 samples.

For all kinds of window_sequences the window_shape of the left half of the first transform window is determined by the window shape of the previous block. The following formula expresses this fact:

$$W_{LEFT,N}(n) = \begin{cases} W_{KBD\_LEFT,N}(n), \text{ if } & window\_shape\_previous\_block == 1 \\ W_{SIN\_LEFT,N}(n), \text{ if } & window\_shape\_previous\_block == 0 \end{cases}$$

where:

window_shape_previous_block: **window_shape** of the block (i-1).

For the first block to be encoded the **window_shape** of the left and right half of the window are identical.

a) ONLY_LONG_SEQUENCE:

The **window_sequence** == ONLY_LONG_SEQUENCE is equal to one LONG_WINDOW (see Table 8.3) with a total window length of 2048.

For **window_shape**==1 the ONLY_LONG_SEQUENCE is given as follows:

$$w(n) = \begin{cases} W_{LEFT,2048}(n), & \text{for } 0 \le n < 1024 \\ W_{KBD\_RIGHT,2048}(n), & \text{for } 1024 \le n < 2048 \end{cases}$$

If **window_shape**==0 the ONLY_LONG_SEQUENCE can be described as follows:

$$w(n) = \begin{cases} W_{LEFT,2048}(n), & \text{for } 0 \le n < 1024 \\ W_{SIN\_RIGHT,2048}(n), & \text{for } 1024 \le n < 2048 \end{cases}$$

The time domain values after windowing ($z_{i,n}$)can be expressed as:

$$z_{i,n} = w(n) \cdot x'_{i,n};$$

b) LONG_START_SEQUENCE:

The LONG_START_SEQUENCE is needed to obtain a correct overlap and add for a block transition from a ONLY_LONG_SEQUENCE to a EIGHT_SHORT_SEQUENCE.

If **window_shape**==1 the LONG_START_SEQUENCE is given as follows:

$$w(n) = \begin{cases} W_{LEFT,2048}(n), & \text{for } 0 \le n < 1024 \\ 1.0, & \text{for } 1024 \le n < 1472 \\ W_{KBD\_RIGHT,256}(n+128-1472), & \text{for } 1472 \le n < 1600 \\ 0.0, & \text{for } 1600 \le n < 2048 \end{cases}$$

If **window_shape**==0 the LONG_START_SEQUENCE looks like:

$$w(n) = \begin{cases} W_{LEFT,2048}(n), & \text{for } 0 \le n < 1024 \\ 1.0, & \text{for } 1024 \le n < 1472 \\ W_{SIN\_RIGHT,256}(n + 128 - 1472), & \text{for } 1472 \le n < 1600 \\ 0.0, & \text{for } 1600 \le n < 2048 \end{cases}$$

The windowed domain values can be calculated with the formula explained in a).

c) EIGHT_SHORT

The **window_sequence** == EIGHT_SHORT comprises eight overlapped SHORT_WINDOWs (see Table 8.3) with a window length of 256 samples each (2048 in total). Each of the eight short windows are windowed separately first. The short window number is indexed with the variable j = 0, …, 7.
The **window_shape** of the previous block influences the first of the eight short windows ($W_0(n)$) only.
If **window_shape**==1 the window functions can be given as follows:

$$W_0(n) = \begin{cases} W_{LEFT,256}(n), & \text{for } 0 \le n < 128 \\ W_{KBD\_RIGHT,256}(n), & \text{for } 128 \le n < 256 \end{cases}$$

$$W_{1-7}(n) = \begin{cases} W_{KBD\_LEFT,256}(n), & \text{for } 0 \le n < 128 \\ W_{KBD\_RIGHT,256}(n), & \text{for } 128 \le n < 256 \end{cases}$$

Otherwise, if **window_shape**==0, the window functions can be described as:

$$W_0(n) = \begin{cases} W_{LEFT,256}(n), & \text{for } 0 \le n < 128 \\ W_{SIN\_RIGHT,256}(n), & \text{for } 128 \le n < 256 \end{cases}$$

$$W_{1-7}(n) = \begin{cases} W_{SIN\_LEFT,256}(n), & \text{for } 0 \le n < 128 \\ W_{SIN\_RIGHT,256}(n), & \text{for } 128 \le n < 256 \end{cases}$$

An EIGHT_SHORT sequence $x'_{i,n}$ is subdivided into eight overlapping segments first and then multiplied with the appropriate window function:

$$z_{i,n} = \begin{cases} x'_{i,n+448} \cdot W_0(n), & \text{for } 0 \le n < 256 \\ x'_{i,n+576} \cdot W_1(n-256), & \text{for } 256 \le n < 512 \\ x'_{i,n+704} \cdot W_2(n-512), & \text{for } 512 \le n < 768 \\ x'_{i,n+832} \cdot W_3(n-768), & \text{for } 768 \le n < 1024 \\ x'_{i,n+960} \cdot W_4(n-1024), & \text{for } 1024 \le n < 1280 \\ x'_{i,n+1088} \cdot W_5(n-1280), & \text{for } 1280 \le n < 1536 \\ x'_{i,n+1216} \cdot W_6(n-1536), & \text{for } 1536 \le n < 1792 \\ x'_{i,n+1344} \cdot W_7(n-1792), & \text{for } 1792 \le n < 2048 \end{cases}$$

d) LONG_STOP_SEQUENCE

This window_sequence is needed to switch from a EIGHT_SHORT_SEQUENCE back to a ONLY_LONG_SEQUENCE.

If **window_shape**==1 the LONG_STOP_SEQUENCE is given as follows:

$$w(n) = \begin{cases} 0.0, & \text{for } 0 \le n < 448 \\ W_{LEFT,256}(n-448), & \text{for } 448 \le n < 576 \\ 1.0, & \text{for } 576 \le n < 1024 \\ W_{KBD\_RIGHT,2048}(n), & \text{for } 1024 \le n < 2048 \end{cases}$$

If **window_shape**==0 the LONG_START_SEQUENCE is determined by:

$$w(n) = \begin{cases} 0.0, & \text{for } 0 \le n < 448 \\ W_{LEFT,256}(n-448), & \text{for } 448 \le n < 576 \\ 1.0, & \text{for } 576 \le n < 1024 \\ W_{SIN\_RIGHT,2048}(n), & \text{for } 1024 \le n < 2048 \end{cases}$$

The windowed time domain values can be calculated with the formula explained in a).

MDCT

The spectral coefficient, $X_{i,k}$, are defined as follows:

$$X_{i,k} = 2 \cdot \sum_{n=0}^{N-1} z_{i,n} \cos\left(\frac{2\pi}{N}(n+n_0)\left(k+\frac{1}{2}\right)\right) \text{ for } 0 \le k < N/2 .$$

where:

z_{in} = windowed input sequence

n = sample index

k = spectral coefficient index

i = block index

N = window length of the one transform window based on the window_ sequence value

n_0 = (N / 2 + 1) / 2

The analysis window length N of one transform window of the mdct is a function of the syntax element **window_sequence** and is defined as follows:

$$N = \begin{cases} 2048, & \text{if ONLY\_LONG\_SEQUENCE (0x0)} \\ 2048, & \text{if LONG\_START\_SEQUENCE (0x1)} \\ 256, & \text{if EIGHT\_SHORT\_SEQUENCE (0x2) (8 times)} \\ 2048, & \text{if LONG\_STOP\_SEQUENCE (0x3)} \end{cases}$$

Diagrams

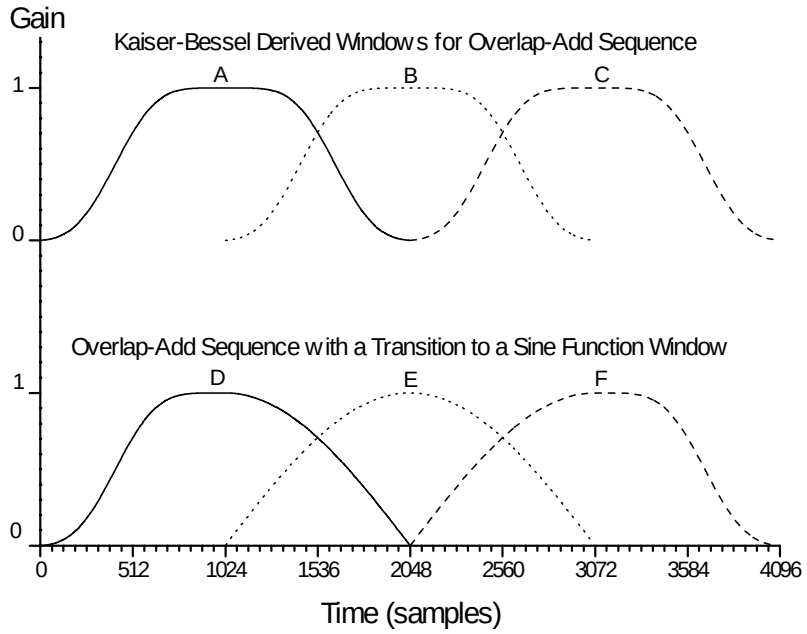


Figure B.2.3.1 -- Example of the Window Shape Adaptation Process.

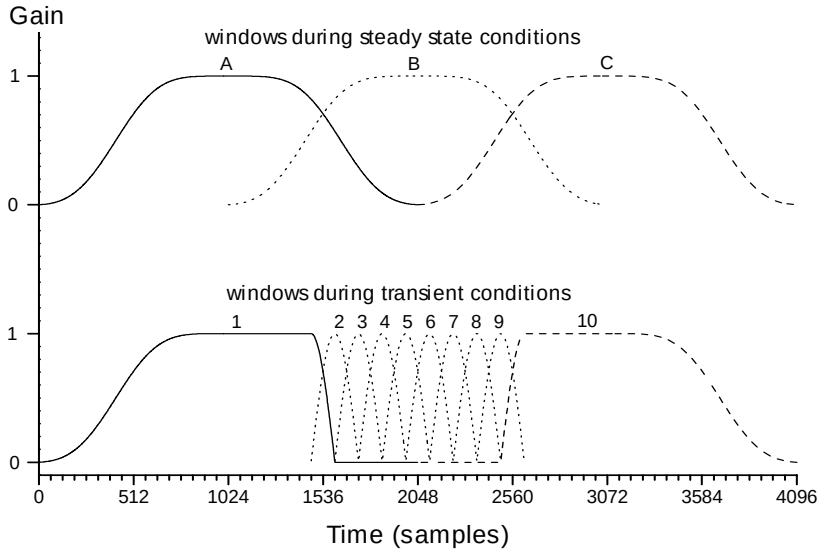


Figure B.2.3.2 – Example of Block Switching During Transient Signal Conditions

Prediction

Tool description

Since each predictor itself is identical on both, the encoder and decoder side, all descriptions and definitions as specified for the decoder in section 13 of the normative part are also valid here.

Prediction is used for an improved redundancy reduction and is especially effective in case of more or less stationary parts of a signal which belong to the most demanding parts in terms of required bitrate. Prediction can be applied to every channel using an intra channel (or mono) predictor which exploits the auto-correlation between the spectral components of consecutive frames. Because a window_sequence of type EIGHT_SHORT_SEQUENCE indicates signal changes, i.e. non-stationary signal characteristic, prediction is only used if window_sequence is of type ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE or LONG_STOP_SEQUENCE.

For each channel prediction is applied to the spectral components resulting from the spectral decomposition of the filterbank. For each spectral component up to limit specified by PRED_SFB_MAX, there is one corresponding predictor resulting in a bank of predictors, where each predictor exploits the auto-correlation between the spectral component values of consecutive frames.

The overall coding structure using a filterbank with high spectral resolution implies the use of backward adaptive predictors to achieve high coding efficiency. In this case, the predictor coefficients are calculated from preceding quantized spectral components in the encoder as well as in the decoder and no additional side information is needed for the transmission of predictor coefficients - as would be required for forward adaptive predictors. A second order backward-adaptive lattice structure predictor is used for each spectral component, so that each predictor is working on the spectral component values of the two preceding frames. The predictor parameters are adapted to the current signal statistics on a frame by frame base, using an LMS based adaptation algorithm. If prediction is activated, the quantizer is fed with a prediction error instead of the original spectral component, resulting in a coding gain.

Encoding process

For each spectral component up to the limit specified by PRED_SFB_MAX of each channel there is one predictor. The following description is valid for one single predictor and has to be applied to each predictor. As said above, each predictor is identical on both, the encoder and decoder side. Therefore, the predictor structure is the same as shown in figure 8.1 and the calculations of the estimate $x_{est}(n)$ of the current spectral component $x(n)$ as well as the calculation and adaptation of the predictor coefficients are exactly the same as those described for the decoder in clause 8.3.2 of the normative part.

The only difference on the encoder side is that the prediction error has to be calculated according to

$$e(n) = x(n) - x_{est}(n)$$

to be fed to the quantizer. In this case the quantized prediction error is transmitted instead of the quantized spectral component.

Predictor control

In order to guarantee that prediction is only used if this results in a coding gain, an appropriate predictor control is required and a small amount of predictor control information has to be transmitted to the decoder. For the predictor control, the predictors are grouped into scalefactor bands.

The following description is valid for either one single_channel_element or one channel_pair_element and has to be applied to each such element. Since prediction is only used if window_sequence is of type ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE or LONG_STOP_SEQUENCE for the channel associated with the single_channel_element or for both channels associated with the channel_pair_element, the following applies only in these cases.

The predictor control information for each frame, which has to be transmitted as side information, is determined in two steps. First, it is determined for each scalefactor band whether or not prediction leads to a coding gain and if yes, the **prediction_used** bit for that scalefactor band is set to one. After this has been done for all scalefactor bands up to PRED_SFB_MAX, it is determined whether the overall coding gain by prediction in this frame compensates at least the additional bit need for the predictor side information. If yes, the **predictor_data_present** bit is set to 1, the complete side information including that needed for predictor reset (see below) has to be transmitted and the prediction error value is fed to the quantizer. Otherwise, the **predictor_data_present** bit is set to 0, the **prediction_used** bits are all reset to zero and are not transmitted. In this case, the spectral component value is fed to the quantizer. Figure B 2.4.1 shows a block diagram of the prediction unit for one scalefactor band. As described above, the predictor control first operates on all predictors of one scalefactor band and is then followed by a second step over all scalefactor bands.

In case of a single_channel_element or a channel_pair_element with **common_window** = 0 the control information is calculated and valid for the predictor bank(s) of the channel(s) associated with that element. In case of a channel_pair_element with **common_window** = 1 the control information is calculated considering both channels associated with that element together. In this case the control information is valid for both predictor banks of the two channels in common.

Predictor reset

When the encoding process is started, all predictors are initialized. This means that the predictor state variables for each predictor are set as follows: $r_0 = r_1 = 0$, $COR_1 = COR_2 = 0$, $VAR_1 = VAR_2 = 1$.

A cyclic reset mechanism is applied by the encoder, where all predictors are initialized again in a certain time interval in an interleaved way. On one hand this increases the stability by re-synchronizing the predictors of the encoder and the decoder and on the other hand it allows defined entry points in the bitstream.

The whole set of predictors is subdivided into 30 so-called reset groups according to the following table:

Reset group number	Predictors of reset group
1	P_0, P_{30}, P_{60}, P_{90},...
2	P_1, P_{31}, P_{61}, P_{91},...
3	P_2, P_{32}, P_{62}, P_{92},...
...	
30	P_{29}, P_{59}, P_{89}, P_{119},...

where P_i is the predictor which corresponds to the spectral coefficient indexed by i.

An encoder is required to reset at least one group every 8 frames and to reset every group within the maximum reset interval of $8 \times 30 = 240$ frames. It is recommended that groups be reset in ascending order, although that is not mandatory. Shorter reset intervals are supported by the bitstream syntax, for example if one group is reset every fourth frame, then all predictors can be reset within an interval of 4 x 30 = 120 frames.

A typical reset cycle starts with reset group number 1 and the reset group number is then incremented by 1 until it reaches 30, and then it starts with 1 again. Nevertheless, it may happen, e.g. due to switching between programs (bitstreams) or cutting and pasting, that there will be a discontinuity in the reset group numbering. If this is the case, there are the following three possibilities for the decoder to operate:

- Ignore the discontinuity and carry on the normal processing. This may result in a short audible distortion due to a mismatch (drift) between the predictors in the encoder and decoder. After one complete reset cycle (reset group n, n+1, ..., 30, 1, 2, ..., n-1) the predictors are re-synchronized again. Furthermore, a possible distortion is faded out because of the attenuation factors a and b.
- Detect the discontinuity, carry on the normal processing but mute the output until one complete reset cycle is performed and the predictors are re-synchronized again.
- Reset all predictors.

The reset mechanism is controlled by the **pred_reset** bit, which always has to be transmitted as soon as the **predictor_data_present** bit is set to 1, and the **pred_reset_group_number**, which specifies the group of predictors to be reset and has only to be transmitted if the **pred_reset** bit is set to 1. If a group reset is applied in the current frame, the **pred_reset** bit has to be set to 1 and the number of the predictor group to be reset has to be coded and transmitted as 5 bit unsigned binary number in **pred_reset_group_number**. After the quantized value has been reconstructed (see below), the reset of the specified group is then carried out by initializing all predictors belonging to that group as described above.

If no group reset is applied in the current frame, the **pred_reset** bit has to be set to 0.

In case of a single_channel_element or a channel_pair_element with **common_window** = 0 the reset has to be applied to the predictor bank(s) of the channel(s) associated with that element and the control information relates to each individual channel. In case of a channel_pair_element with **common_window** = 1 the reset has to be applied to the two predictor banks of the two channels associated with that element and the control information relates to both channels together.

In case of short blocks, i.e. window_sequence is of type EIGHT_SHORT_SEQUENCE, prediction is always disabled and a reset is carried out for all predictors in all scalefactor bands, which is equivalent to a complete reinitialization, see above.

Reconstruction of the quantized spectral component

Since the reconstructed value of the quantized spectral component is needed as predictor input signal, it has to be calculated in the encoder, see also figure 13.2 in the normative part and figure B.2.4.1. Depending on the value of the **prediction_used** bit, the reconstructed value is either the quantized spectral component or the quantized prediction error. Therefore, the following steps are necessary:

- If the bit is set (1), then the quantized prediction error, reconstructed from data to be transmitted, is added to the estimate $x_{est}(n)$, calculated by the predictor, resulting in the reconstructed value of the quantized spectral component, i.e.
  $$x_{rec}(n) = x_{est}(n) + e_q(n)$$
- If the bit is not set (0), then the quantized value of the spectral component is identical to the value reconstructed directly from the data to be transmitted.

Diagrams

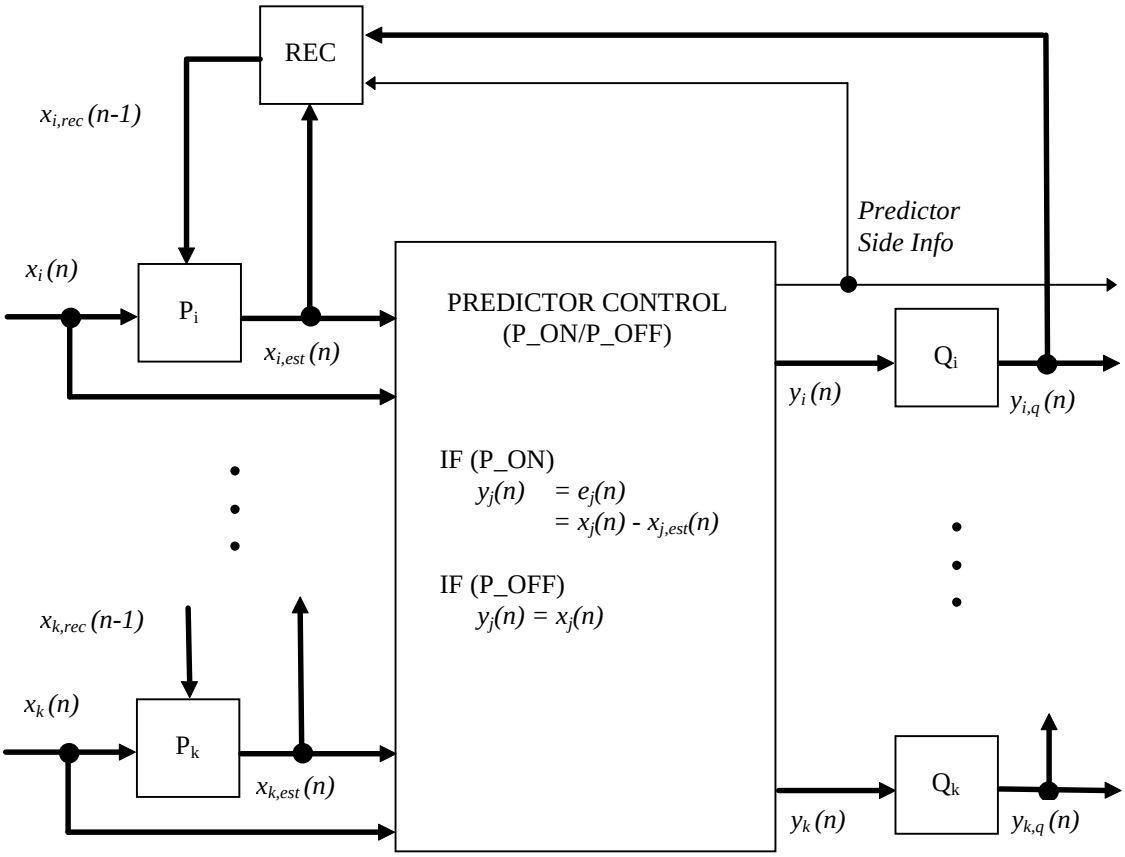


Figure B.2.4.1 -- Block diagram of prediction unit for one scalefactor band. The complete processing is only shown for predictor P_i (Q - quantizer, REC - reconstruction of last quantized value). Note that the predictor control operates on all predictors P_i ... P_j ... P_k of a scalefactor band and is followed by a second control over all scalefactor bands.

Long Term Prediction

The general structure of the perceptual audio coder with LTP is shown in Figure 1. First, the optimal LTP is estimated. Then, the predicted signals based quantized data are obtained and corresponding error signals are calculated. Finally, the error signals are quantized and transmitted with the side information, such as predictor control and predictor parameters.

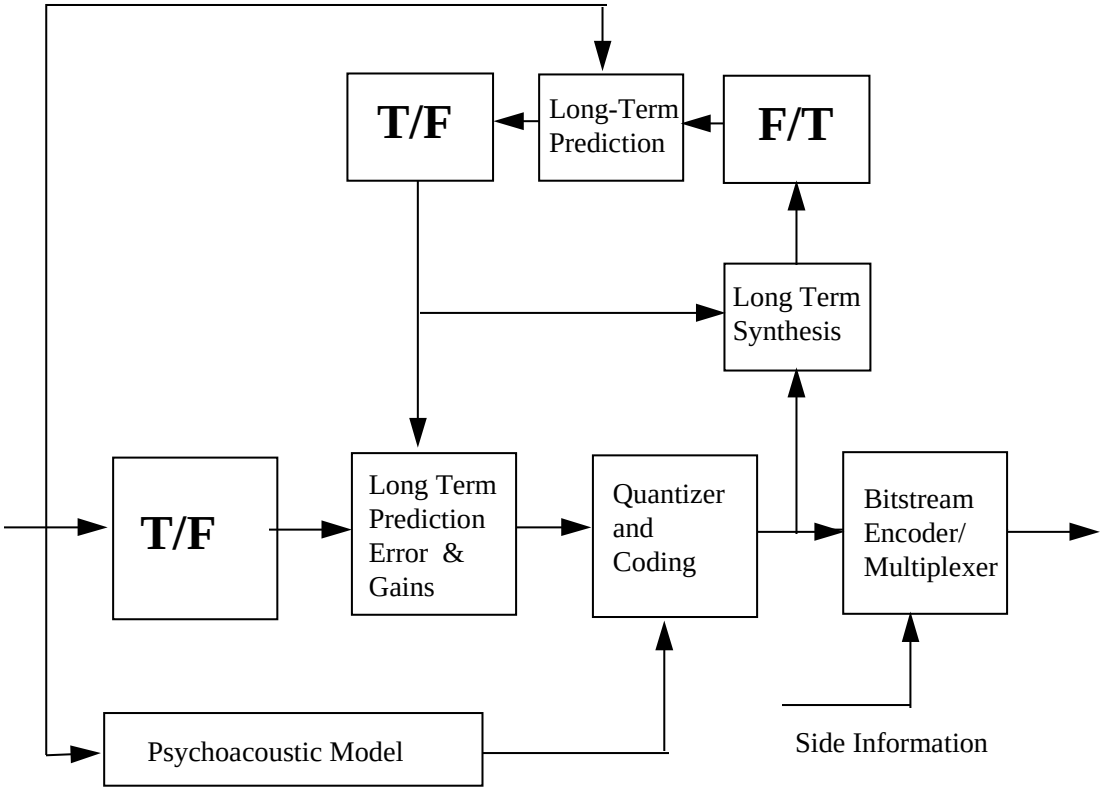


Figure 1. Block diagram of the encoding process of Long Term Prediction

As the perceptual audio coder processes the signal frame by frame, it is obvious that when we want to quantize $\mathbf{X}_{m+1}$, the quantized data $\tilde{\mathbf{X}}_k, k = m, m-1, m-2, \ldots,$ are available both in encoder and decoder. In time domain, it means that the quantised time domain samples $\tilde{x}(i), i = mN, mN-1, \ldots,$ are available.

To remove the redundancy of the signal in present frame $m+1$ based on the previously quantized data, an LTP is used

$$P(z) = \sum_{k=-m_1}^{m_2} b_k z^{-(\alpha+k)}$$

where α represents a long delay in the range 1 to 1024. For $m_1 = m_2 = 0$, we have a one-tap predictor.

The parameters α and b_k are determined by minimizing the mean squared error over the whole window. The delay α is quantized with 10 bits with 1024 possible values in the range 1 to 1024. For prediction coefficients b_k are non-uniformly quantized to 3 bits. Due to their nonuniform distribution, nonuniform quantization has to be used. The table for coding the coefficient values is presented in the decoding process description.

After the LTP is determined, the predicted signal can be obtained for the $(m+1)$th frame

$$\hat{x}(i) = \sum_{k=-m_1}^{m_2} b_k \tilde{x}(i - 2N + 1 - k - \alpha),$$

$$i = mN + 1, mN + 2, \ldots, (m+1)N$$

Using the forward MDCT, we have the predicted spectral coefficient $\tilde{\mathbf{X}}_{m+1}$ for the $(m+1)$th frame. In order to guarantee that prediction is only used if it results in a coding gain, an appropriate predictor control is required and a small amount of predictor control information has to be transmitted to the decoder. The predictor control scheme is the same as the backward predictor control scheme which has been used in MPEG-2 Advanced Audio Coding (AAC), i.e., one bit per scalefactor band is sent as side information to flag whether prediction is used for that band.

In the encoding process the prediction gain resulting from LTP is estimated for each scalefactor band by comparing the predicted energy to the amount of side information needed. LTP is switched on only for those scalefactor bands where there is

positive net coding gain. The corresponding bits of side information are set accordingly. Finally, the net coding gain for the whole frame (all scalefactor bands) is checked, and the switch for using LTP at all for the frame is set accordingly.

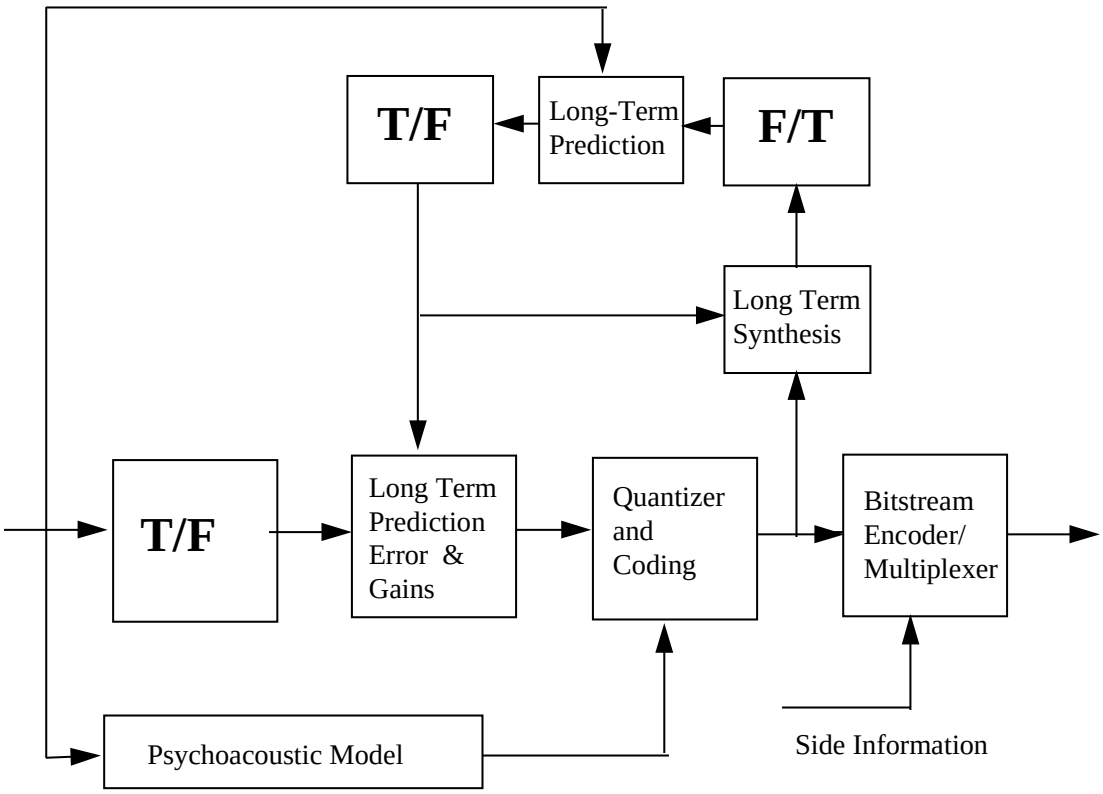


Figure 1. Block diagram of the encoding process of Long Term Prediction

Temporal Noise Shaping (TNS)

Temporal Noise Shaping is used to control the temporal shape of the quantization noise within each window of the transform. This is done by applying a filtering process to parts of the spectral data of each channel.
Encoding is done on a window basis. The following steps are carried out to apply the Temporal Noise Shaping tool to one window of spectral data:

- A target frequency range for the TNS tool is chosen. A suitable choice is to cover a frequency range from 1.5 kHz to the uppermost possible scalefactor band with one filter. Please note that this parameter (TNS_MAX_BANDS) depends on profile and sampling rate as indicated in the normative part.

- Next, a linear predictive coding (LPC) calculation is carried out on the spectral MDCT coefficients corresponding to the chosen target frequency range. For better stability, coefficients corresponding to frequencies below 2.5 kHz may be excluded from this process. Standard LPC procedures as known from speech processing can be used for the LPC calculation, e.g. the well-known Levinson-Durbin algorithm. The calculation is carried out for the maximum permitted order of the noise shaping filter (TNS_MAX_ORDER). Please note that this value depends on the profile as indicated in the normative part.

- As a result of the LPC calculation, the expected prediction gain g_p is known as well as the TNS_MAX_ORDER reflection coefficients r[] (so-called PARCOR coefficients).

- If the prediction gain g_p does not exceed a certain threshold t, no temporal noise shaping is used. In this case, the tns_data_present bit is set to zero and TNS processing is finished. A suitable threshold value is t = 1.4.

- If the prediction gain g_p exceeds the threshold t, temporal noise shaping is used.

- In a next step the reflection coefficients are quantized using coef_res bits. An appropriate choice for coef_res is 4 bits. The following pseudo code describes the conversion of the reflection coefficients r[] to index values index[] and back to quantized reflection coefficients rq[].

```
iqfac   = ((1 << (coef_res-1)) - 0.5) / (π/2.0);
iqfac_m = ((1 << (coef_res-1)) + 0.5) / (π/2.0);

/* Reflection coefficient quantization */
for (i=0; i<TNS_MAX_ORDER; i++)  {
    index[i] = NINT( arcsin( r[i] ) * ((r[i] >= 0) ? iqfac : iqfac_m) );
}
/* Inverse quantization */
for (i=0; i<TNS_MAX_ORDER; i++)  {
    rq[i] = sin( index[i] / ((index[i] >= 0) ? iqfac : iqfac_m) );
}
```

where arcsin() denotes the inverse sin() function.

- The order of the used noise shaping filter is determined by subsequently removing all reflection coefficients with an absolute value smaller than a threshold p from the "tail" of the reflection coefficient array. The number of the remaining reflection coefficients is the order of the noise shaping filter. A suitable threshold for truncation is p = 0.1.

- The remaining reflection coefficients rq[] are converted into order+1 linear prediction coefficients a[] (known as "step-up procedure"). A description of this procedure is provided in the normative part as a part of the tool description (see "/* Conversion to LPC coefficients */").

- The computed LPC coefficients a[] are used as the encoder noise shaping filter coefficients. This FIR filter is slid across the specified target frequency range exactly the way it is described in the normative part for the decoding process (tool description). The difference between the decoding and encoding filtering is that the all-pole (auto-regressive) filter used for decoding is replaced by its inverse all-zero (moving-average) filter, i.e. replacing the decoder filter equation

- ```
  y[n]  =  x[n] - a[1]*y[n-1] - ... - a[order]*y[n-order]
  ```

  by the inverse (encoder) filter equation

  ```
  y[n]  =  x[n] + a[1]*x[n-1] + ... + a[order]*x[n-order]
  ```

  By default, an upward direction of the filtering is appropriate.

- Finally, the side information for Temporal Noise Shaping is transmitted:

Bitstream Element	Algorithmic Variable or Value
n_filt	1
coef_res	coef_res-3
coef_compress	0
length	Number of processed scalefactor bands
direction	0 (upwards)
order	Order of noise shaping filter
coef[]	index[]

Joint Coding

M/S Stereo

The decison to code left and right coefficients as either left + right (L/R) or mid/side (M/S) is made on a noiseless coding band by noiseless coding band basis for all spectral coefficients in the current block. For each noiseless coding band the following decison process is used:

1. For each noiseless coding band, not only L and R raw thresholds, but also M=(L+R)/2 and S=(L-R)/2 raw thresholds are calculated. For the raw M and S thresholds, rather than using the tonality for the M or S threshold, one uses the more tonal value from the L or R calculation in each threshold calculation band, and proceed with the psychoacoustic model for M and S from the M and S energies and the minimum of the L or R values for C(ω) in each threshold calculation band. The
  ```

values that are provided to the imaging control process are identified in the psychoacoustic model information section as en(b) (the spread normalized energy) and nb(b), the raw threshold.

2. The raw thresholds for M, S, L and R, and the spread energy for M, S, L and R, are all brought into an "imaging control process". The resulting adjusted thresholds are inserted as the values for cb(b) into step 11 of the psychoacoustic model for further processing.
3. The final, protected and adapted to coder-band thresholds for all of M,S,L and R are directly applied to the appropriate spectrum by quantizing the actual L, R, M and S spectral values with the appropriate calculated and quantized threshold.
4. The number of bits actually required to code M/S, and the number of bits required to code L/R are calculated.
5. The method that uses the least bits is used in each given noiseless coding band, and the stereo mask is set accordingly.

With these definitions

Mthr,Sthr,Rthr, Lthr	raw thresholds. (the nb(b) from step 10 of the psychoacoustic model)
Mengy,Sengy,Rengy,Sengy	spread energy.(en(b) from step 6 of the psychoacoustic model)
Mfthr, Sfthr, Rfthr, Lfthr	final (output) thresholds. (returned as nb(b) in step 11 of the psychoacoustic model)
bmax(b)	BMLD protection ratio, as can be calculated from

$$bmax(b) = 10^{-3\cdot\left[0.5+0.5\cdot\cos\left(\pi\cdot\frac{\min(bval(b),15.5)}{15.5}\right)\right]}$$

the imaging control process for each noiseless coding band is as follows:

```
t=Mthr/Sthr
if (t>1)
        t=1/t
Rfthr= max(Rthr*t, min (Rthr,  bmax*Rengy)
Lfthr= max(Lthr*t, min (Lthr, bmax) *Lengy)
t=min(Lthr, Rthr)
Mfthr=min(t, max(Mthr, min(Sengy*bmax,Sthr) )
Sfthr=min(t, max(Sthr, min(Mengy*bmax,Mthr) )
```

Intensity Stereo Coding

Intensity stereo coding is used to exploit irrelevance in the between both channels of a channel pair in the high frequency region. The following procedure describes one possible implementation while several different implementations are possible within the framework of the defined bitstream syntax.

Encoding is done on a window group basis. The following steps are carried out to apply the intensity stereo coding tool to one window group of spectral data:

- A suitable approach is to code a consecutive region of scalefactor bands in intensity stereo technique starting above a lower border frequency f_0. An average value of f_0 = 6 kHz is appropriate for most types of signals.

- For each scalefactor band, the energy of the left, right and the sum channel is calculated by summing the squared spectral coefficients, resulting in values $E_l[sfb]$, $E_r[sfb]$, $E_s[sfb]$. If the window group comprises several windows, the energies of the included windows are added.

- For each scalefactor band, the corresponding intensity position value is computed as

$$is\_position[sfb] = NINT\left(2\cdot\log_2\left(\frac{E_l[sfb]}{E_r[sfb]}\right)\right)$$

- Next, the intensity signal spectral coefficients $spec_i[i]$ are calculated for each scalefactor bands by adding spectral samples from the left and right channel ($spec_l[i]$ and $spec_r[i]$) and rescaling the resulting values like

$$spec_i[i] = (spec_l[i] + spec_r[i])\cdot\sqrt{\frac{E_l[sfb]}{E_s[sfb]}}$$

- The intensity signal spectral components are used to replace the corresponding left channel spectral coefficients. The corresponding spectral coefficients of the right channel are set to zero.

Then, the standard process for quantization and encoding is performed on the spectral data of both channels. However, the prediction status of the right channel predictors is forced to "off" for the scalefactor bands coded in intensity stereo. These predictors are updated by using an intensity decoded version of the quantized spectral coefficients. The procedure for this is described in the tool description for the intensity stereo decoding process in the normative part.

Finally, before transmission the Huffman codebook INTENSITY_HCB (15) is set in the sectioning information for all scalefactor bands that are coded in intensity stereo.

Quantization

Introduction

The description of the AAC quantization module is subdivided into three levels. The top level is called "loops frame program". The loops frame program calls a subroutine named "outer iteration loop" which calls the subroutine "inner iteration loop". For each level a corresponding flow diagram is shown.

The loops module quantizes an input vector of spectral data in an iterative process according to several demands. The inner loop quantizes the input vector and increases the quantizer step size until the output vector can be coded with the available number of bits. After completion of the inner loop an outer loop checks the distortion of each scalefactor band and, if the allowed distortion is exceeded, amplifies the scalefactor band and calls the inner loop again.

AAC loops module input:
1. vector of the magnitudes of the spectral values mdct_line(0..1023).
2. xmin(sb) (see B 2.1.4. „Steps in threshold calculation“, Step 12)
3. mean_bits (average number of bits available for encoding the bitstream).
4. more_bits, the number of bits in addition to the average number of bits, calculated by the psychoacoustic module out of the perceptual entropy (PE).
5. the number and width of the scalefactor bands (see table 3.5 normative part)
6. for short block grouping the spectral values have to be interleaved so that spectral lines that belong to the same scalefactor band but to different block types which shall be quantized with the same scalefactors are put together in one (bigger) scalefactor band (for a full description of grouping see clause 3.3.4 normative part)

AAC loops module output:
1. vector of quantized values x_quant(0..1023).
2. a scalefactor for each scalefactor band (sb)
3. common_scalefac (quantizer step size information for all scalefactor bands)
4. number of unused bits available for later use.

Preparatory steps

Reset of all iteration variables

1. The start value of gloabal_gain for the quantizer is calculated so that all quantized MDCT values can be encoded in the bitstream :.

$start\_common\_scalefac = 16/3*(\log_2((max\_mdct\_line \wedge (3/4))/MAX\_QUANT)$

max_mdct_line is the maximum MDCT coefficient, MAX_QUANT is the maximum quantized value witch can be encoded in the bitstream, it is defined to 8191. During the iteration process, the common_scalefac must not become less than start_common_scalefac.

2. All scalefactors are set to zero.

Bit reservoir control

Bits are saved to the reservoir when fewer than the mean_bits are used to code one frame.

*mean_bits = bit_rate * 1024 / sampling_rate.*

The number of bits which can be saved in the bit reservoir at maximum is called 'maximum_bitreservoir_size' which is calculated using the procedure outlined in normative clause 3.2.2. If the reservoir is full, unused bits have to be encoded in the bitstream as fillbits.

The maximum amount of bits available for a frame is the sum of mean_bits and bits saved in the bit reservoir.

The number of bits that should be used for encoding a frame depends on the more_bits value which is calculated by the psychoacoustic model and the maximum available bits. The simplest way to control bit reservoir is :

if more_bits > 0 :
available_bits = average_bits + min (more_bits, bitres_bits)
if more_bits < 0 :
available_bits = average_bits + max (more_bits, bitres_bits - maximum_bitreservoir_size)

Quantization of MDCT coefficients

The formula for the quantization in the encoder is the inverse of the decoder dequatization formula (see also the decoder description) :

*x_quant = int ((abs(mdct_line) * (2^(¼ * (sf_decoder - SF_OFFSET))))^(3/4) + MAGIC_NUMBER)*

MAGIC_NUMBER is defined to 0.4054, SF_OFFSET is defined as 100 and mdct_line is one of spectral values, which is calculated from the MDCT. These values are also called 'coefficients'

For use in the iteration loops, the scalefactor 'sf_decoder' is split in two variables:

sf_decoder = scalefactor - common_scalefac + SF_OFFSET

It follows from this, that the formula used in the distortion control loop is:

*x_quant = int (abs(mdct_line) * (2^(¼ * (scalefactor - common_scalefac))))^(3/4) + MAGIC_NUMBER)*

The sign of the *mdct_line* is saved separately and added again only for counting the bits and encoding the bitstream

Outer iteration loop (distortion control loop)

The outer iteration loop controls the quantization noise which is produced by the quantization of the frequency domain lines within the inner iteration loop. The coloring of the noise is done by multiplication of the lines within scalefactor bands with the actual scalefactors before doing the quantization. The following pseudo-code illustrates the multiplication.

*do for each scalefactor band **sb**:*
*do from lower index to upper index **i** of scalefactor band*
*mdct_scaled(i) = abs(mdct_line(i))^(3/4) * 2^(3/16 * (scalefactor(sb))*
end do
end do

Call of inner iteration loop

For each outer iteration loop (distortion control loop) the inner iteration loop (rate control loop) is called. The parameters are the frequency domain values with the scalefactors applied to the values within the scalefactor bands (mdct_scaled(0..1023)), a start value for common_scalefac, and the number of bits which are available to the rate control loop. The result is the number of bits actually used and the quantized frequency lines x_quant(i), and a new common_scalefac.
The Formula to calculate the quantized MDCT coefficients is:

*x_quant(i) = int ((mdct_scaled (i) * 2^(-3/16 * common_scalefac)) + MAGIC_NUMBER)*

The bits, that would be needed to encode the quantizes values and the side information (scalefactors etc.) are counted according to the bitstream syntax, described in [A 2.8 Noisless Coding].

Amplification of scalefactor bands which violate the masking threshold

The calculation of the distortion (*error_energy(sb)*)of the scalefactor band is done as follows:

```
do for each scalefactor band sb:
    error_energy(sb)=0
    do from lower index to upper index i of scalefactor band
        error_energy(sb) = error_energy(sb) + (abs( mdct_line(i))
                    - (x_quant(i) ^(4/3) * 2^( -¼ * (scalefactor(sb) -common_scalefac ))))^2
    end do
end do
```

All spectral values of the scalefactor bands which have a distortion that exceeds the allowed distortion (*xmin(sb)*)are amplified according to formula in B 2.7.4.1 (“Outer Iteration Loop“), the new scalefactors can calculated according to this pseudocode:

```
do for each scalefactor band sb
        if ( error_energy(sb) > xmin(sb) ) then
                scalefactor(sb) = scalefactor(sb) + 1
        end if
end do
```

Conditions for the termination of the loops processing

Normally the loops processing terminates, if there is no scalefactor band with more than the allowed distortion. However this is not always possible to obtain. In this case there are other conditions to terminate the outer loop. If

- All scalefactor bands are already amplified, or
- The difference between two consecutive scalefactors is greater than 64

The loop processing stops, and by restoring the saved scalefactors(sb) a useful output is available. For real-time implementation, there might be a third condition added which terminates the loops in case of a lack of computing time.

Inner iteration loop (rate control loop)

The inner iteration loop calculates the actual quantization of the frequency domain data (*mdct_scaled*) with the following function, which uses the formula from B.2.7.4.2: „call of Inner iteration loop“ :

```
quantize_spectrum(x_quant[] , mdct_scaled[] , common_scalefac):
        do for all MDCT coefficients i :
                x_quant(i) = int (( mdct_scaled (i) * 2^(-3/16 * common_scalefac)) + MAGIC_NUMBER)
        end do
```

and then calls a function *bit_count()*. This function counts the number of bits that would be neccessary to encode a bitstream according to clause 1 “Syntax“ of the normative part.

The inner iteration loop can be implemented using successive approximation.:

```
inner_loop():
        if (outer_loop_count == 0 )
```

```
            common_scalefac = start_common_scalefac
            quantizer_change = 64
    else
            quantizer_change = 2
    end if
    do
            quantize_spectrum()
            counted_bits = bit_count()
            quantizer change = quantizer_change / 2
            if (counted_bits > available_bits) then
                    common_scalefac = common_scalefac + quantizer_change
            else
                    common_scalefac = common_scalefac - quantizer_change
            end if
            if (quantizer_change == 0) && (counted_bits > available_bits)
                    quantizer_change = 1
            end if
    while ( quantizer_change != 0 )
```

Due to the choice of *start_common_scalefac* calculated form 2.7.2.1, after the first run through the inner loop the number of needed bits is always greater than the available bits , and therefor *common_scalefac* is always increased by the *quantizer_change.*

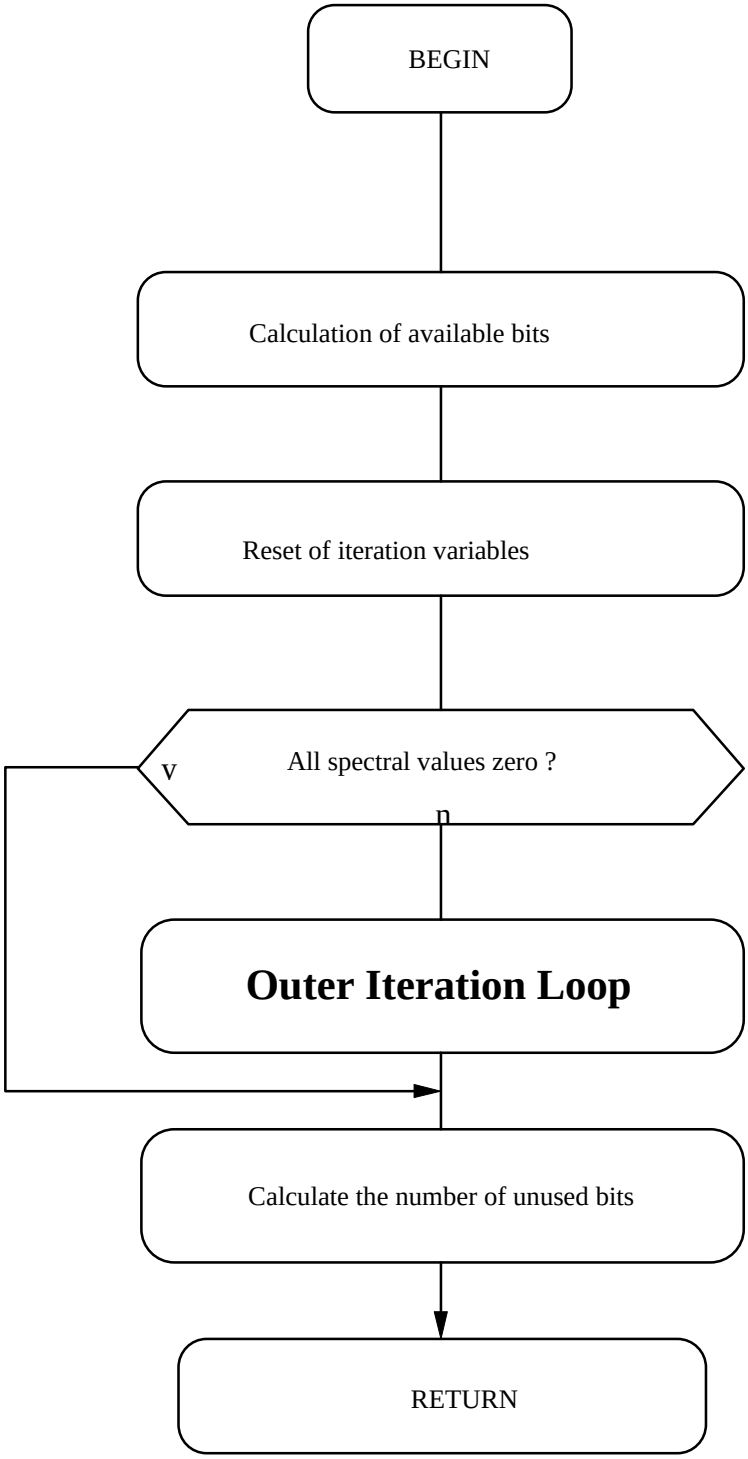


Figure B.2.7.1 -- AAC iteration loop

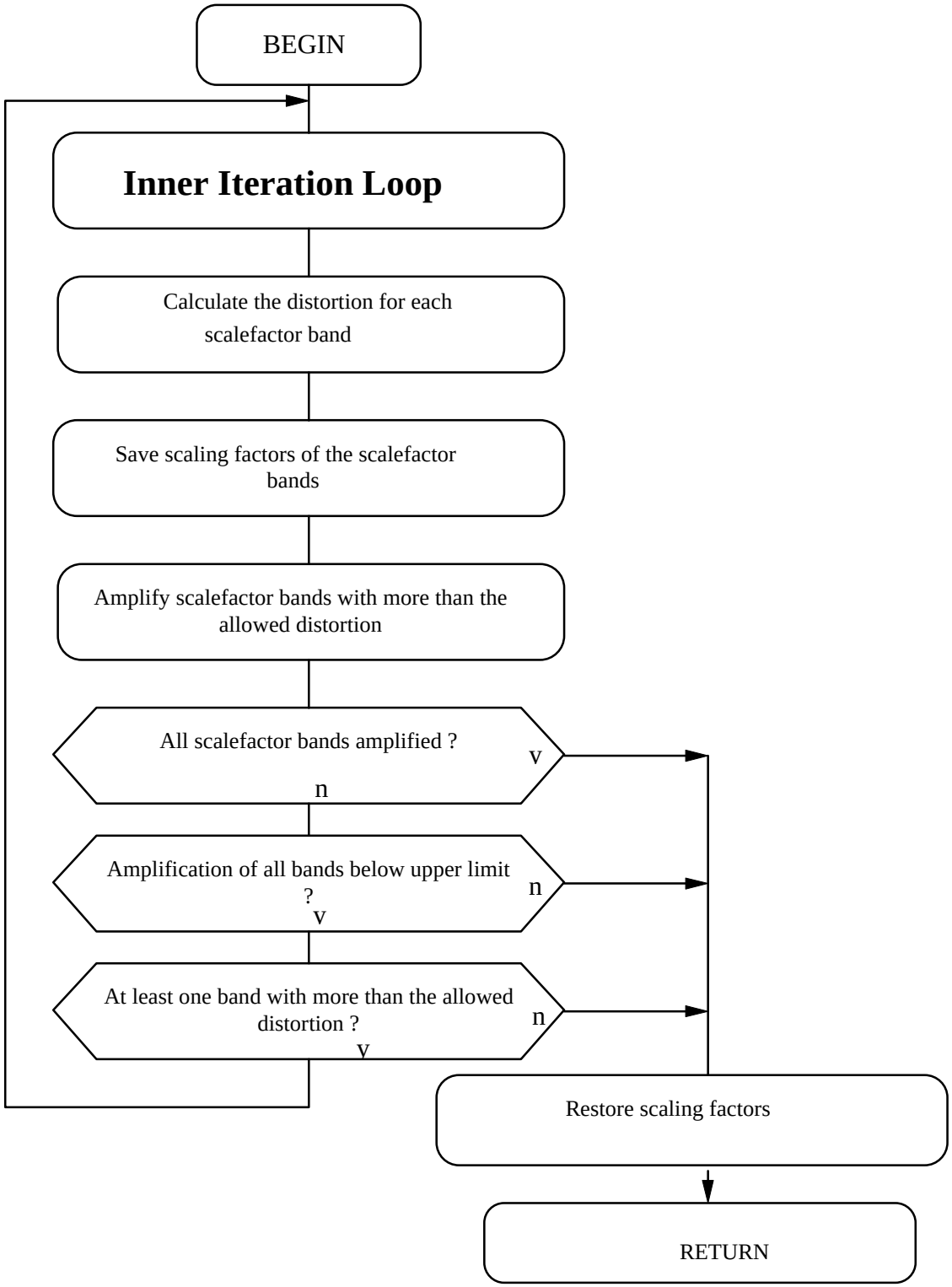


Figure B.2.7.2 -- AAC outer iteration loop

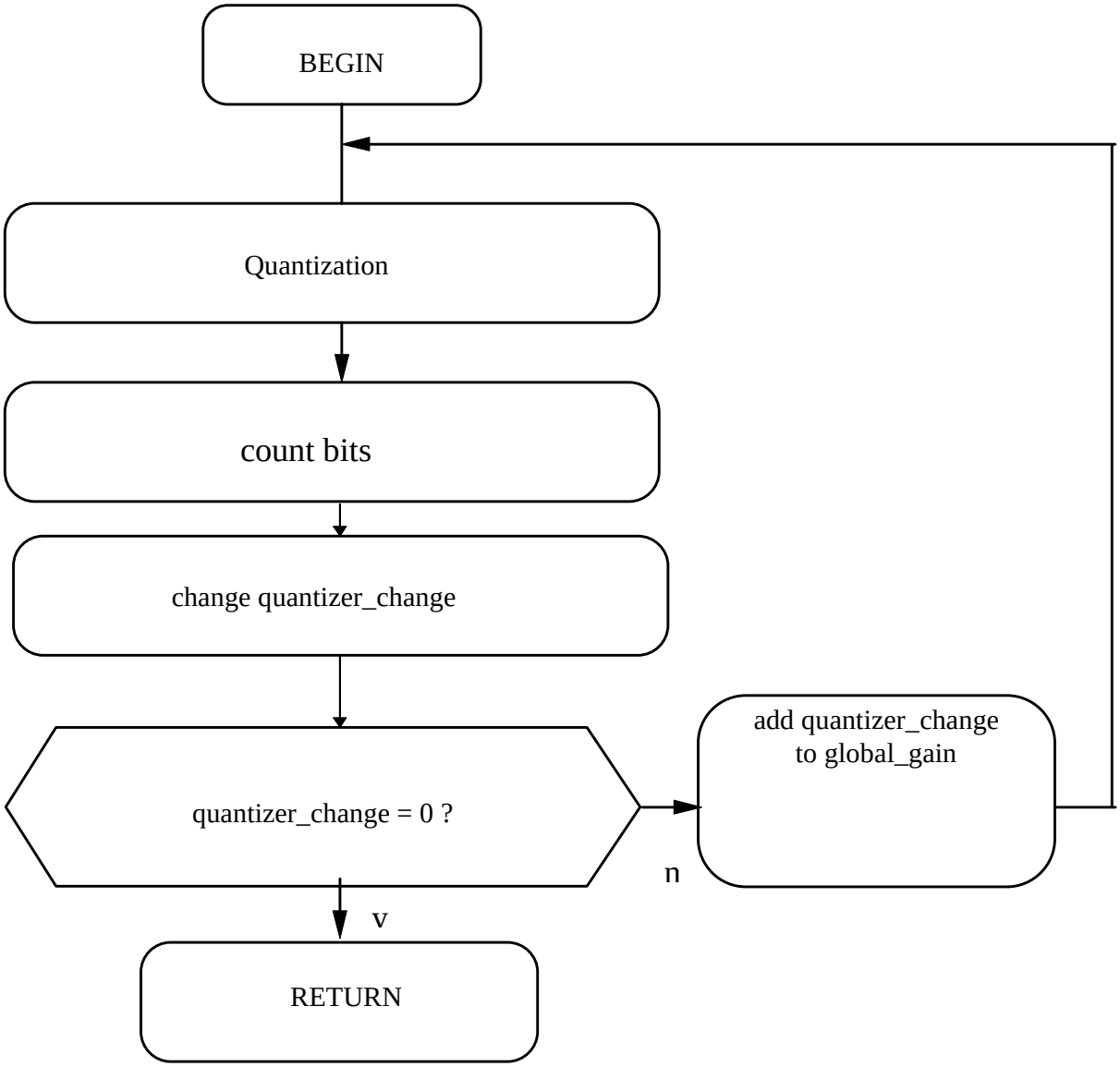


Figure B.2.7.3 -- AAC inner iteration loop

Noiseless Coding

Introduction

In the AAC encoder the input to the noiseless coding module is the set of 1024 quantized spectral coefficients. Since the noiseless coding is done inside the quantizer inner loop, it is part of an iterative process that converges when the total bit count (of which the noiseless coding is the vast majority) is within some interval surrounding the allocated bit count. This section will describe the encoding process for a single call to the noiseless coding module.

Noiseless coding is done via the following steps:

- Spectrum clipping
- Preliminary Huffman coding using maximum number of sections
- Section merging to achieve lowest bit count

Spectrum clipping

As a first step a method of noiseless dynamic range limiting may be applied to the spectrum. Up to four coefficients can be coded separately as magnitudes in excess of one, with a value of +-1 left in the quantized coefficient array to carry the sign. The index of the scalefactor band containing the lowest-frequency “clipped” coefficients is sent in the bitstream. Each of the “clipped” coefficients is coded as a magnitude (in excess of 1) and an offset from the base of the previously indicated scalefactor band. For this the long block scalefactor bands and coefficient ordering within those bands are used regardless of the window sequence. One strategy for applying spectrum clipping is to clip high-frequency coefficients whose absolute

amplitudes are larger than one. Since the side information for carrying the clipped coefficients costs some bits, this noiseless compression is applied only if it results in a net savings of bits.

Sectioning

The noiseless coding segments the set of 1024 quantized spectral coefficients into *sections*, such that a single Huffman codebook is used to code each section (the method of Huffman coding is explained in a later section). For reasons of coding efficiency, section boundaries can only be at scalefactor band boundaries so that for each section of the spectrum one must transmit the length of the section, in scalefactor bands, and the Huffman codebook number used for the section.

Sectioning is dynamic and typically varies from block to block, such that the number of bits needed to represent the full set of quantized spectral coefficients is minimized. This is done using a greedy merge algorithm starting with the maximum possible number of sections each of which uses the Huffman codebook with the smallest possible index. Sections are merged if the resulting merged section results in a lower total bit count, with merges that yield the greatest bit count reduction done first. If the sections to be merged do not use the same Huffman codebook then the codebook with the higher index must be used.

Sections often contain only coefficients whose value is zero. For example, if the audio input is band limited to 20 kHz or lower, then the highest coefficients are zero. Such sections are coded with Huffman codebook zero, which is an escape mechanism that indicates that all coefficients are zero and it does not require that any Huffman codewords be sent for that section.

Grouping and interleaving

If the window sequence is eight short windows then the set of 1024 coefficients is actually a matrix of 8 by 128 frequency coefficients representing the time-frequency evolution of the signal over the duration of the eight short windows. Although the sectioning mechanism is flexible enough to efficiently represent the 8 zero sections, *grouping* and *interleaving* provide for greater coding efficiency. As explained earlier, the coefficients associated with contiguous short windows can be grouped such that they share scalefactors amongst all scalefactor bands within the group. In addition, the coefficients within a group are interleaved by interchanging the order of scalefactor bands and windows. To be specific, assume that before interleaving the set of 1024 coefficients *c* are indexed as

c[g][w][b][k]

where

g is the index on groups
w is the index on windows within a group
b is the index on scalefactor bands within a window
k is the index on coefficients within a scalefactor band
and the right-most index varies most rapidly.

After interleaving the coefficients are indexed as

c[g][b][w][k]

This has the advantage of combining all zero sections due to band-limiting within each group.

Scalefactors

The coded spectrum uses one quantizer per scalefactor band. The step sizes of each of these quantizers is specified as a set of scalefactors and a global gain which normalizes these scalefactors. In order to increase compression, scalefactors associated with scalefactor bands that have only zero-valued coefficients are ignored in the coding process and therefore do not have to be transmitted. Both the global gain and scalefactors are quantized in 1.5 dB steps. The global gain is coded as an 8-bit unsigned integer and the scalefactors are differentially encoded relative to the previous scalefactor (or global gain for the first scalefactor) and then Huffman coded. The dynamic range of the global gain is sufficient to represent full-scale values from a 24-bit PCM audio source.

Huffman coding

Huffman coding is used to represent n-tuples of quantized coefficients, with the Huffman code drawn from one of 11 codebooks. The spectral coefficients within n-tuples are ordered (low to high) and the n-tuple size is two or four coefficients. The maximum absolute value of the quantized coefficients that can be represented by each Huffman codebook and the number of coefficients in each n-tuple for each codebook is shown in Table B.2.8.1. There are two codebooks for each maximum absolute value, with each representing a distinct probability distribution function. The best fit is always chosen. In order to

save on codebook storage (an important consideration in a mass-produced decoder), most codebooks represent unsigned values. For these codebooks the magnitude of the coefficients is Huffman coded and the sign bit of each non-zero coefficient is appended to the codeword.

Table B.2.8.1 -- Huffman Codebooks

Codebook index	n-Tuple size	Maximum absolute value	Signed values
0		0	
1	4	1	yes
2	4	1	yes
3	4	2	no
4	4	2	no
5	2	4	yes
6	2	4	yes
7	2	7	no
8	2	7	no
9	2	12	no
10	2	12	no
11	2	16 (ESC)	no

Two codebooks require special note: codebook 0 and codebook 11. As mentioned previously, codebook 0 indicates that all coefficients within a section are zero. Codebook 11 can represent quantized coefficients that have an absolute value greater than or equal to 16. If the magnitude of one or both coefficients is greater than or equal to 16, a special *escape coding* mechanism is used to represent those values. The magnitude of the coefficients is limited to no greater than 16 and the corresponding 2-tuple is Huffman coded. The sign bits, as needed, are appended to the codeword. For each coefficient magnitude greater or equal to 16, an *escape sequence* is also appended, as follows:

escape sequence = <escape_prefix><escape_separator><escape_word>

where
<escape_prefix> is a sequence of N binary “1’s”
<escape_separator> is a binary “0”
<escape_word> is an N+4 bit unsigned integer, msb first
and N is a count that is just large enough so that the magnitude of the quantized coefficient is equal to

2^(N+4) + <escape_word>

Perceptual Noise Substitution (PNS)

The encoding procedure for noise substitution is similar to the encoding procedure for intensity stereo and is performed as follows:

- For each scalefactor band containing spectral coefficients above a lower border frequency (e.g. 4 kHz) a noise detection is carried out. The scalefactor band is classified as noise-like if the corresponding signal is neither tonal nor contains strong changes in energy over time. The tonality of the signal can be estimated by using the tonality values calculated in the psychoacoustic model. Similarly, changes in signal energy can be evaluated using the FFT energies calculated in the psychoacoustic model.
- From the detection procedure, a map, noise_flag[sfb], is constructed such that noise-like scalefactor bands are flagged with a non-zero value.
- For each flagged scalefactor band the energy (sum of squares) of the corresponding spectral coefficients is calculated and mapped to a logarithmic representation with a resolution of 1.5 dB. An offset (NOISE_OFFSET=90) is added to the logarithmic noise energy values.
- For each flagged scalefactor band, the corresponding spectral coefficients are set to zero before quantization of the coefficients is carried out as usual.
- During the noiseless coding procedure, the pseudo-codebook NOISE_HCB is set for all flagged scalefactor bands. Apart from this, the regular section / noiseless coding procedure is carried out on the quantized coefficient data.

- The logarithmic noise energy values are coded analogous to the regular scalefactors, i.e. with a differential encoding scheme starting with the „global_gain value“. They are transmitted in place of the scalefactors belonging to the flagged scalefactor bands.

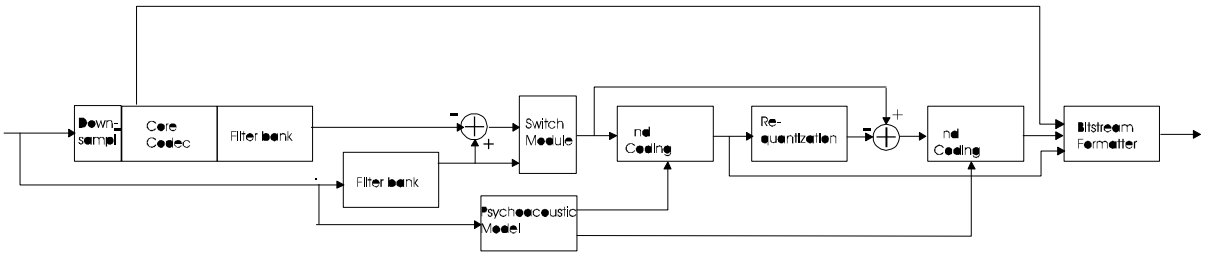
Scalable AAC with Core Coder

Mono or Stereo Encoder

The description provided below describes one possible way of achieving bit rate scalability. Bit-rate functionality can also be achieved using individual coding schemes as well. Here, a particular configuration is described where the T/F modules and a core coder is used. Since it is based on the calculation of a difference signal, the core coder must encode the waveform of the input signal. Currently the tool has been successfully tested based on the ITU-T G.729 codec. A slightly different implementation of the tool was used in the 1995 MPEG-4 test with the FS 1016 CELP coder and the „individual lines“ parametric coder of the MPEG-4 audio VM.

In addition to the core coder there is at least one enhancement layer based on the T/F based VM modules. In order to allow for integer frame lengths, depending on the core coder, alternative frame lengths for the AAC module may be used. The T/F modules provides, in addition to the standard 1024 length, also a 960 samples per frame implementation, which at 48 kHz sampling rate leads to 20 ms and at 32 kHz to a frame length of 30 ms. This allows for an easy integration of the MPEG-4 VM CELP core, and to construct bitstream frames which integrate standard speech coders, like G.729, which have a frame length of a multiple of 10 ms.

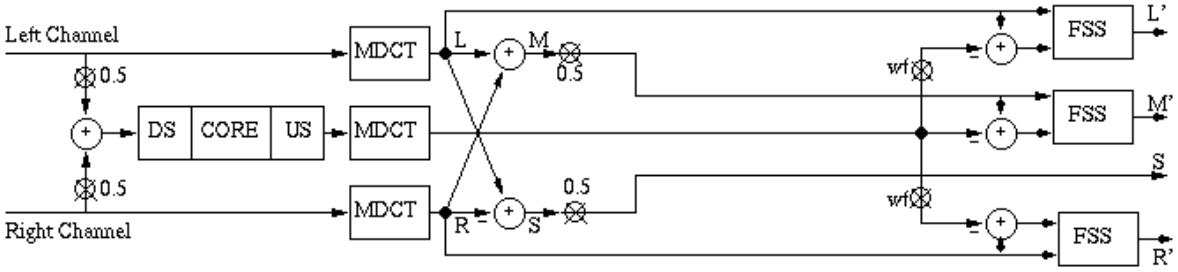
The bitrate of the additional layers can be any bit rate defined for the T/F based enhancement stages. The ratio of the sampling rate of the core coder and the sampling rate of the enhancement coder must be an integer.



Encoder structure

Mixed Mono/Stereo Coder

The block diagrams of the encoders of mode 1 and mode 2 are given in Figure 1 and 2. The block diagram of the third mode is identical to mode 2, with the core coder path



removed. Fig. 1: Core + stereo T/F

From the stereo input signal a mono signal is derived, which is then coded/decoded with a core coder, which - in general - operates at a lower sampling rate, up-sampled and transformed into the frequency-

domain. This part is identical to the corresponding part of the mono scalable coder. Next the standard M/S-Stereo matrix, as defined for the AAC coder is calculated on the spectra of the Left (L) and Right (R) signal, giving the Mid (M) and Side (S) signals. Three Frequency-Selective Switch modules (FSS) - identical to the switch modules of the mono scalable coder - then are used to generate the signals L', R' and M' from the L, R, and M signals. These signals and the original S signal then are used as the input signal to the standard AAC joint stereo coding modules. Both, M/S- and Intensity stereo coding is possible. The standard AAC ms_mask_present and ms_used[] flags are used to indicate M/S or L/R coding. However, since either M/S or L/R coding is used, it is not necessary to transmit all three FSS-related side-information. Instead, if M/S coding is selected for a band, only the FSS information for the M-switch needs to be transmitted in diff_control[0][win][sfb]. diff_control[1][win][sfb] is not transmitted in this case. If L/R coding is selected the FSS-side-information for the L and R channel needs to be transmitted in diff_control[0][win][sfb] for the left channel and in diff_control[1][win][sfb] for the right channel.

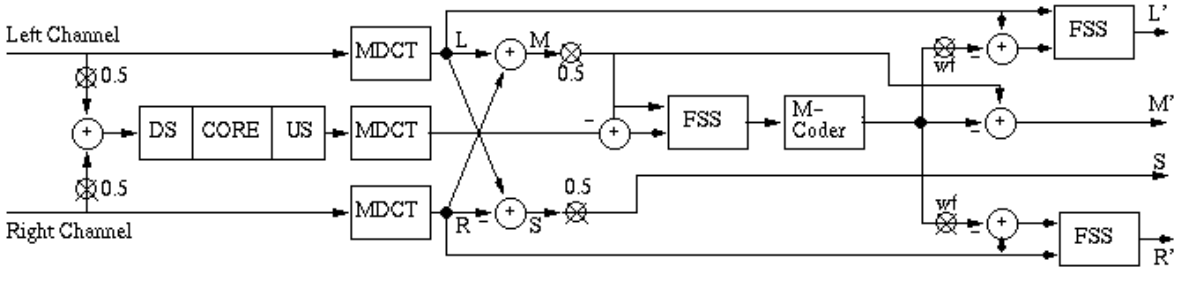


Fig. 2: Core + mono T/F + stereo T/F

This is an extended version of the Core + stereo T/F coder. An additional T/F - Coder is used to encode the output signal of the core-coder in the frequency domain. This makes the core plus M - T/F coder combination identical to the scalable mono core plus T/F coder. Two more FSS modules allow to select L'/R' to be identical to L, or R, respectively, or L - M*wf or R-M*wf. Again some of the switching conditions need not to be transmitted: If M/S coding is selected , only the FSS information for the M-switch needs to be transmitted in diff_control[0][win][sfb]. Diff_control[1][win][sfb] is not transmitted.. If L/R coding is selected the FSS-side-information for the L and R and the M channel needs to be transmitted in diff_control[0][win][sfb] for the left channel and in diff_control[1][win][sfb] for the right channel, and diff_control_m[win][0] for the M channel.

Bit-Sliced Arithmetic Coding (BSAC)

In BSAC, a quantized sequence is mapped into a bit-sliced sequence as shown in Figure bsac1.

MSB LSB

$B_{0,m}$	$B_{0,m-1}$	…	$B_{0,0}$	0^{th} quantized spectral data
$B_{1,m}$	$B_{1,m-1}$	…	$B_{1,0}$	
$B_{2,m}$	$B_{2,m-1}$	…	$B_{2,0}$	
…	…	…	…	
$B_{k,m}$	$B_{k,m-1}$	…	$B_{k,0}$	k^{th} quantized spectral data

where, allocated_bit is (m+1) bit

⇒

$vector_{0,m}$ … $vector_{k/4,m}$

$B_{0,m}$	$B_{1,m}$	$B_{2,m}$	$B_{3,m}$	…	$B_{k-3,m}$	$B_{k-2,m}$	$B_{k-1,m}$	$B_{k,m}$

$vector_{0,m-1}$ $vector_{k/4,m-1}$

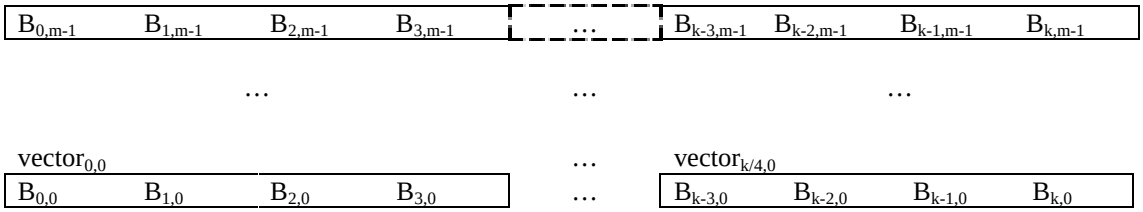


Figure bsac1. Bit-Sliced Representation

In order to maximize the match of the statistics of the bit-sliced sequences to that of the arithmetic model, 4-dimension vectors are formed from the bit-sliced sequence of the quantized spectrum and the 4-dimension vector is divided into two subvectors depending upon the previous state.
For example, consider a quantized sequence, x[n] as follows :

x[0] = 9, x[1]=0, x[2]=7 and x[3]=11 ...

If the allocated_bit is 5, 5 vectors are formed from a quantized sequence as shown in Figure bsac2.

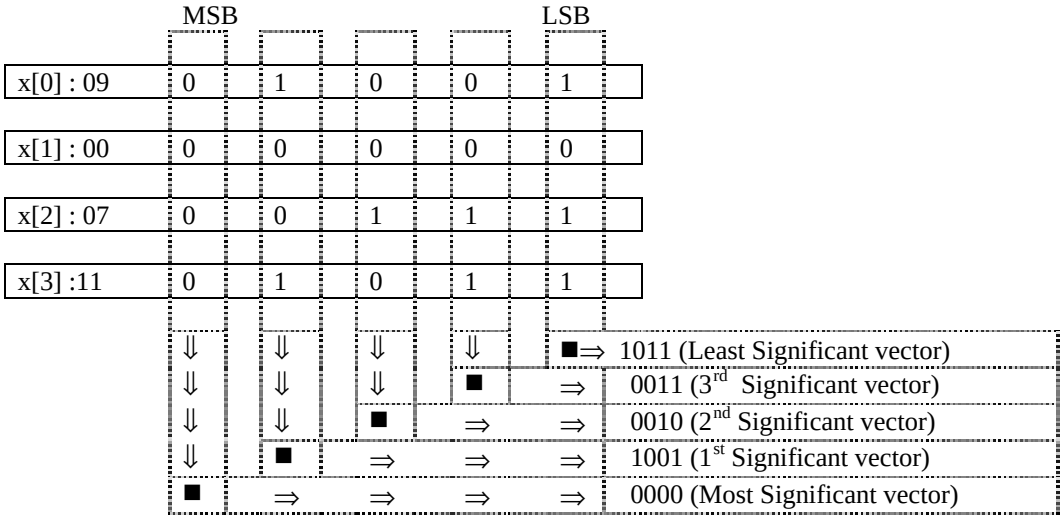
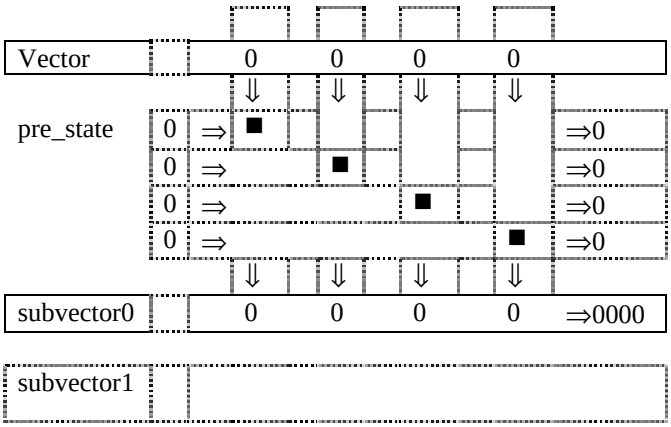
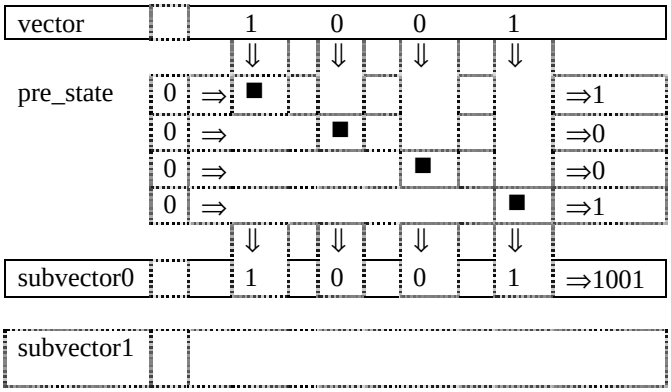


Figure bsac2 Example of Bit-Sliced Vector Representation

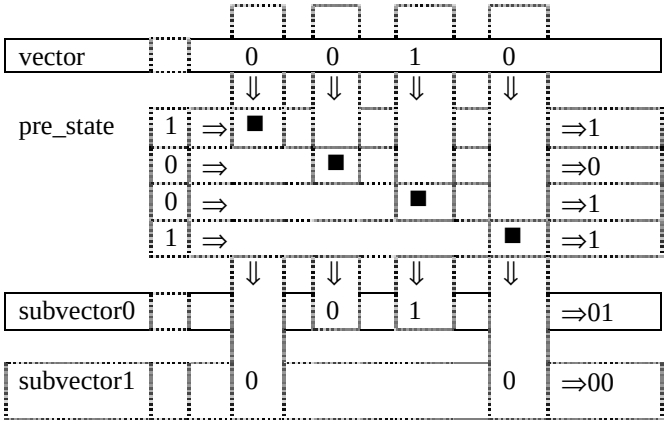
The previous states are updated along with coding the vectors from MSB to LSB. They are initialized to 0, are unchanged when bit value is zero and are set to 1 when bit value is non-zero as shown in Figure bsac3. The vectors are divided into two subvectors as shown in Figure bsac3. One is the subvector which is composed of bit-values whose previous state is 0, another is the subvector which is composed of bit-values whose previous state is 1.



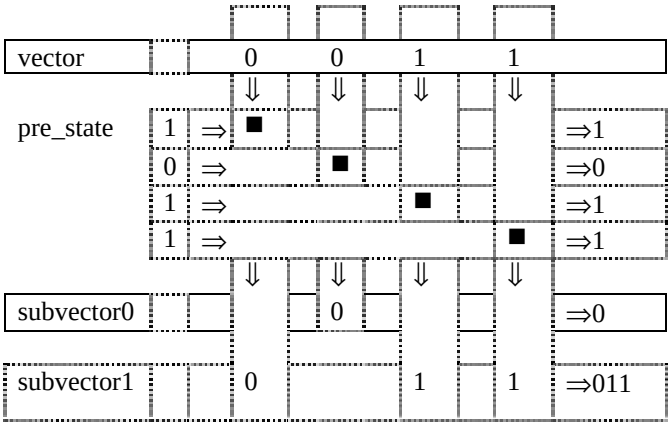
a) Most Significant vector and subvector



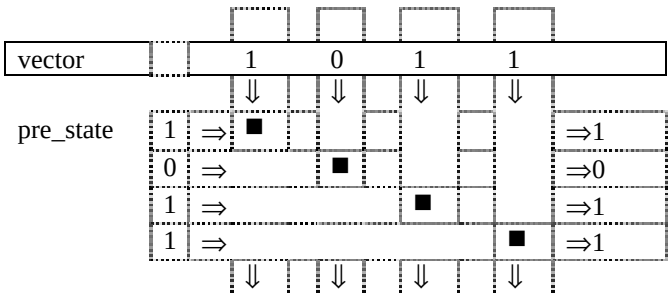
b) 1st Significant vector and subvector

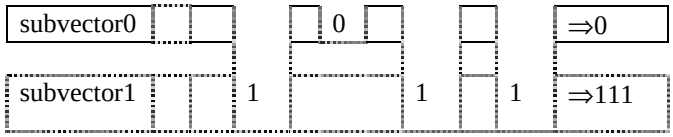


c) 2nd Significant vector and subvector



d) 3rd Significant vector and subvectors





e) Least Significant vector and subvectors

Figure bsac3 Examples of splitting vectors into subvectors

The arithmetic model relies on the allocated_bit of the coding band, the significance of the bit-sliced vector and the previous state.

Scaleable controller

This tool controlls the spectrum normalization and interleave vector quantization tools to construct a scalable coder. In each scaleable layer, spectrum normalization and interleaved vector quantization tool is cascaded. Each scalable layer has active frequency band which is shifted according to signal characteristics.

ANNEX B: Interface definitions for the VM software

Decoder functions

Quantization, Scalefactors, Noiseless Coding

```
void aac_decode_init();                          /* Initial AAC-decoder settings                          */

int aac_decode_frame(                            /* Decode bits to MDCT coefficients                      */
                BsBitStream *fixed_stream,       /*  Input --- bitstream                                  */
                double *spectral_line_vector[MAX_TIME_CHANNELS],  /*  Output spectrum*/
                int *block_type,                 /* Output --- block (window) type                        */
                Window_shape *window_shape);     /* Output --- window shape                               */
```

Interleaved Vector Quantization and Spectrum Normalization

```
void ntt_tf_decoder(                             /* time/frequency mapping decoder                        */
                ntt_INDEX  *indexp,              /* input --- index packet                                */
                double out[]);                   /* output --- time domain reconstructed signal           */

void ntt_tf_proc_spectrum_d(                     /* de-normalizer of flattened MDCT coefficients          */
                ntt_INDEX  *indexp,              /* input --- index packet                                */
                double flat_spectrum[],          /* input --- decoded flattened MDCT                      */
                double spectrum[]);              /* output --- denormalized MDCT                          */

void ntt_tf_freq2time(                           /*  frequency to time conversion IMDCT                   */
                double spectrum[],               /* input --- MDCT coefficient                            */
                int    w_type,                   /* input --- window type                                 */
                double audio_sample[]);          /* output --- time domain reconstructed signal           */

void ntt_dec_lpc_spectrum(                       /* LPC spectrum decoder                                  */
                int    index_lsp[ntt_N_SUP_MAX][ntt_LSP_NIDX_MAX],   /* in LSP index   */
                int    w_type,                   /* input --- window type                                 */
                double lpc_spec[]);              /* output --- decoded LPC spectrum (square root)         */

void ntt_dec_bark_env(                           /* Bark-scale envelope decoder                           */
                int    index_fw[],               /* input --- index of Bark-scale envelope VQ             */
                int    index_fw_alf[],           /* input --- index of prediction switch                  */
                int    w_type,                   /* input --- window type                                 */
                int    pf_switch,                /* input --- postfilter switch                           */
                double bark_env[]);              /* output --- reconstructed Bark-scale envelope          */

void ntt_dec_gain(                               /* global and subframegain decoder                       */
                int    index_pow[],              /* input --- index  of global and subframe power         */
                int    w_type,                   /* input --- window type                                 */
                double gain[]);                  /* output --- decoded power                              */
```

```
void ntt_denormalizer_spectrum(          /* denormalize MDCT coefficients                  */
                int    w_type,           /* Input --- window type                          */
                double flat_spectrum[],  /* Input --- reconstructed flattened MDCT         */
                double gain[],           /* Input --- reconstructed values of subframe gain*/
                double pit_seq[],        /* Input --- reconstructed pitch components       */
                double ggain[],          /* Input --- reconstructed value of global gain:  */
                double bark_env[],       /* Input --- reconstructed Bark-scale envelope    */
                double lpc_spec[],       /* Input --- reconstructed LPC envelope           */
                double spectrum[]);      /* Output --- denormalized MDCT coefficients      */

void ntt_post_process(                   /*  post process for MDCT coefficients            */
                int    index_pf,         /* Input --- switch for postfilter                */
                int    w_type,           /* Input --- window type                          */
                double spectrum[],       /* Input --- reconstructed MDCT coefficients      */
                double lpc_spectrum[],   /* Input --- reconstructed LPC spectrum           */
                double out_spectrum[]);  /* Output --- post processed MDCT coefficients    */


void ntt_dec_pitch(                      /*  decoder of pitch components                   */
                int    index_pit[],      /* Input --- index of pitch frequency             */
                int    index_pls[],      /* Input --- index of pitch components shape VQ   */
                int    index_pgain[],    /* Output --- index of pitch gain                 */
                int    w_type,           /* Output --- window type                         */
                double pit_seq[],        /* Output --- locally decoded pitch components    */
                double pgain[]);         /* Output --- locally decoded value of pitch gain */

void ntt_dec_pgain(                      /* decoder of pitch gain                          */
                int    index,            /* Input --- index of pitch gain                  */
                double *pgain);          /* Output --- reconstructed value of pitch gain   */

void ntt_dec_pit_seq(                    /* decoder of pitch component shape VQ            */
                int    index_pit[],      /* Input --- index of pitch frequency             */
                int    index_pls[],      /* Input --- index of pitch coponents shape       */
                double pit_seq[]);               /* Output --- reconstructed pitch components      */

void ntt_vec_lenp(                       /* Pitch vec. bits for each divided subvector     */
                int bits[],              /* output --- bits assigned for pitch subvector   */
                int length[]);           /* output --- dimension of  pitch subvector       */

void ntt_extend_pitch(                   /* extend pitch components onto MDCT coefficients */
                int    index_pit[],/* Input --- index of pitch frequency                  */
                double  pit_pack[],      /* Input --- paccked array of pitch components    */
                double  pit_seq[]);      /* Output --- extended array onto MDCT coefficients*/


void ntt_dec_lpc_spectrum_inv(           /* decoder of reciprocal LPC envelope             */
                int    index_lsp[ntt_N_SUP_MAX][ntt_LSP_NIDX_MAX], /* Input LSP index*/
                int    w_type,           /* Input --- window type                          */
                double inv_lpc_spec[]);  /* Output --- reconstructed reciprocal LPC env.   */

void ntt_TfInit(                         /* initializer T/F mapping coder                  */
                float  sampling_rate,    /* input --- sampling rate                        */
                float  bit_rate,         /* input --- bit rate                             */
                long   num_channel,      /* input --- number of channel                    */
                int    *frameNumSample,  /* input --- number of sample per frame           */
                int    *delayNumSample,  /* input --- number of delay sample          */
                int    *med_win_in_long, /* input --- number of medium subframe            */
                int    *short_win_in_long);/* input --- number of short subframe           */
```

```
int ntt_BitUnPack(                      /* unpack bitstream                                  */
                BsBitStream *stream,    /* input --- bitstream                               */
                int    available_bits,  /* input --- available bits                          */
                int    block_type,      /* input --- block (window) type                     */
                ntt_INDEX *index,       /* output ---- index packet                          */
                int    InitFlag);       /* input ---- init flag                              */

void ntt_win_sw_init(                   /* initializer for sin window                        */
                int    max_ch,          /* input ---maximan channel number                   */
                int    block_size_samples, /* input --- number of samples in block (window) */
                int    sampling_rate,   /* input --- sampling rate                           */
                int    bit_rate,        /* input --- bitrate                                 */
                int    short_win_in_long); /* input ---                                     */


void ntt_init();                        /* initial setting                                   */

void ntt_cp_mdtbl(                      /* read mode table                                   */
                ntt_MODE_TABLE ltbl);


void ntt_vec_len(                       /* get subvector lengtht (dimension) and bits        */
                int    bits[],          /* output --- assigned bits for two channel code     */
                int    length[]);       /* output --- dimension (length) of subvectors       */

void ntt_vec_lens(                      /* get subvector lengtht (dimension) and bits        */
                int    bits[],          /* output --- assigned bits for two channel code     */
                int    length[]);       /* output --- dimension (length) of subvectors       */

void ntt_vec_lenm(                      /* get subvector lengtht (dimension) and bits        */
                int    bits[],          /* output --- assigned bits for two channel code     */
                int    length[]);       /* output --- dimension (length) of subvectors       */


void ntt_vex_pns(                       /*  reconstructor from codebook (short frame)        */
                int    *index,          /* index --- index for MDCT coefficients             */
                double *sig);           /* index --- reconstructed flattened MDCT            */

void ntt_vex_pnm(                       /*  reconstructor from codebook (medium frame)       */
                int    *index,          /* index --- index for MDCT coefficients             */
                double *sig);           /* index --- reconstructed flattened MDCT            */

oid ntt_get_cdbk(                               /* read codebook file                        */
                char   *name,           /* input --- codebook name                           */
                double *codev_l,        /* output --- reconstruction code vector             */
                int    cb_size,         /* input --- codebook size (1<<numbits)              */
                int    cb_len,          /* input --- dimension of reconstruction vector      */
                int    cb_len_mx);      /* input --- maximam codebook dimension              */

void ntt_lsptowts(                      /* LSP to reciprocal LPC envelope (short frame)      */
                double lsp[],           /* input --- LSP parameters                          */
                double wt[]);           /*output --- reciprocal LPC envelope                 */

void ntt_fwdecl(                        /* decoder of Bark-scale envelope (long frame)       */
                int    index[],         /* input --- index for Bark-scale envelope           */
                int    ind_alf,         /* input --- index for prediction coeff in Bark-scale */
                int    pf_switch,       /* input --- switch for postfilter                   */
                int    i_sup,           /* input --- channel number                          */
                double pred[],          /* output --- reconstructed envelope                 */
                int    InitFlag);       /* input --- init flag                               */
```

```
void ntt_fwdecm(                        /* decoder of Bark-scale envelope (medium frame)   */
                int    index[],         /* input --- index for Bark-scale envelope         */
                int    ind_alf,         /* input --- index for prediction coeff in Bark-scale  */
                int    pf_switch,       /* input --- switch for postfilter                 */
                int    i_sup,           /* input --- channel number                        */
                double pred[],          /* output --- reconstructed envelope               */
                int    InitFlag);       /* input --- init flag                             */

void ntt_fwdecs(                        /* decoder of Bark-scale envelope (short frame)    */
                int    index[],         /* input --- index for Bark-scale envelope         */
                int    ind_alf,         /* input --- index for prediction coeff in Bark-scale  */
                int    i_sup,           /* input --- channel number                        */
                double pred[],          /* output --- reconstructed envelope               */
                int    InitFlag);       /* input --- init flag                             */

void ntt_fwexs(                         /* look up codebook        (short mode)            */
                int    index[],         /* input --- index for Bark-scale envelope         */
                double env[]);          /* outpur --- code vector output                   */

void ntt_fwexm(                         /* look up codebook        (medium frame )         */
                int    index[],         /* input --- index for Bark-scale envelope         */
                double env[]);          /* outpur --- code vector output                   */

void ntt_fwex(                          /* look up codebook        (long frame)            */
                int    index[],         /* input --- index for Bark-scale envelope         */
                double env[]);          /* outpur --- code vector output                   */

void ntt_cnst_chk(                      /* check constant if it is beyond ranges           */
                int    calcv,           /* input --- constant valure                  */
                int    defv,            /* input --- predetermined value                   */
                char   *defname);       /* input --- constant name                         */


void ntt_lsptowd(                       /* LSP to LPC envelope with interpolation (long)   */
                double lsp[],           /* input --- reconstruction LSP parameter          */
                double wt[]);           /* output --- LPC spectral envelope                */

void ntt_lsptowmd(                      /* LSP to LPC envelope with interpolation (medium)*/
                double lsp[],           /* input --- reconstruction LSP parameter          */
                double wt[]);           /* output --- LPC spectral envelope                */

void ntt_adjust_bits(                   /* assign bits                                     */
                int    available_bits,  /* input --- avalable bits                         */
                int    w_type);         /* input --- window type                           */

void ntt_set_interleave(                /* set interleave matrix                           */
                enum   ntt_INTERLEAVE_SET_MODE set_mode);

void ntt_dec_pit_seq(                   /* decoder of pitch components sequence            */
                int    index_pit[],     /* input --- index for pitch frequency             */
                int    index_pls[],     /* input --- index for pitch components shape vq   */
                double pit_seq[])       /* output --- reconstructed pitch components       */

void ntt_get_code(                      /* read LSP structured codebook                    */
                char *fname,            /* input --- codebook  name for LSP                */
                int nstage,             /* input --- number of stage                       */
                int csize[],            /* input --- codebook size for each stage          */
                int cdim[],             /* input --- dimension boundary of code vector     */
                double code[][ntt_N_PR_MAX],    /*output --- codebook array                */
```

```
                double fgcode[ntt_N_MODE_MAX][ntt_MA_NP][ntt_N_PR_MAX]) /*pred   */

void ntt_redec(                         /*  decoder of LSP                                  */
                int ifr_fi,             /* input --- frame number                           */
                int index[],            /* input --- index for LSP                          */
                int nstg1,              /* input --- number of stage                        */
                int csize1[],           /* input --- codebook size for LSP                  */
                int cdim1[],            /* input --- dimension boundary of code vector      */
                double code[][ntt_N_PR_MAX],    /* input --- codebook                       */
                double fgcode[ntt_N_MODE_MAX][ntt_MA_NP][ntt_N_PR_MAX], /*pred   */
                double fg_sum[ntt_N_PR_MAX],  /* input --- contribution from previous       */
                double pred_vec[ntt_N_PR_MAX],          /* input --- previous contribution  */
                double lspq[],          /* output --- reconstructed LSP                     */
                double out_vec[])               /* output --- partially reconstructed LSP for next fr.   */

void ntt_relspwed(                      /* intermediate LSP quantizer                       */
                double lsp[ntt_N_PR_MAX+1],     /* input --- original LSP                   */
                double wegt[ntt_N_PR_MAX+1],  /* input --- weight vector                    */
                double lspq[ntt_N_PR_MAX+1],   /* output --- reconstructed LSP vector       */
                int    ifr_fi,          /* input --- frame number                           */
                int    nstage,          /* input --- number of stage                        */
                int    csize[],         /* input --- codebook size                          */
                int    cdim[],          /* input --- code vector dimension                  */
                double code[][ntt_N_PR_MAX],    /* input --- codebook                       */
                double fg_sum[ntt_N_MODE_MAX][ntt_N_PR_MAX],       /* input pred            */
                double amp_cur,         /* input  --- current frame power                   */
                double cmpe[ntt_N_MODE_MAX][ntt_N_PR_MAX],         /* input buffer          */
                double target_buf[ntt_N_MODE_MAX][ntt_N_PR_MAX],   /* input buffer          */
                double out_vec[ntt_N_PR_MAX], /*output --- reconstructed LSP vector        */
                int    *vqword,         /* output --- packed index for LSP                  */
                int    index_lsp[])     /* output --- index for LSP                         */

void ntt_setd(                          /*  set value                                       */
                int n, double c,        /* input --- scalar value                           */
                double xx[])            /* output --- output data array                     */

void ntt_set_isp(                       /* set baundary for LSP code vector                 */
                int nsp)                /* input --- number of split for LSP code vector    */

void ntt_vq_coder(                      /* t/f encoder based on NTT_VQ mode                 */
                double in[],            /* input --- time domain input signal               */
                double     *spectral_line_vector[MAX_TIME_CHANNELS],          /* input: spectrum*/
                double     external_pw[],  /* input --- external perceptual model           */
                int        pw_select,   /* input --- perceptual model select                */
                int        block_type,  /* input --- block (window) type                    */
                ntt_INDEX   *index,     /* input --- index packet                           */
                ntt_PARAM   *param_ntt,/* input --- parameter packet                        */
                int        available_bits)  /* input --- available bits per frmae           */

void ntt_vq_decoder(                    /*  t/f decoder based on NTT_VQ                     */
                ntt_INDEX  *indexp,     /* input --- index packet                           */
                double *spectral_line_vector[MAX_TIME_CHANNELS])    /* out                  */

void ntt_zerod(                         /* clear array                                      */
                int n,                  /* input --- number of data                 */
                double xx[])            /* in/out --- array                                 */
```

Filterbank Tool

```
void freq2buffer(                       /* Frequency to time conversion IMDCT                */
                double p_in_data[],     /* Input ---- MDCT coefficients                      */
                double p_out_data[],    /* Output --- time domain reconstructed signal       */
                double p_overlap[],     /* Input/Output --- overlap-buffer                   */
                enum WINDOW_TYPE block_type,        /* Input --- window type             */
                int nlong,              /* Input --- shift length for long windows           */
                int nmed,               /* Input --- shift length for medium windows         */
                int nshort,             /* Input --- shift length for short windows          */
                Window_shape wfun_select,       /* Input --- window shape                */
                Imdct_out overlap_select);/* IMDCT output, non-/overlapped, for gain-control */

void imdct(                             /* IMDCT                                             */
                double in_data[],       /* Input --- MDCT Coefficients                       */
                double out_data[],      /* Output --- Time domain reconstructed signal       */
                int len);               /* Input --- Length of the MDCT coefficients-vector  */

void calc_window(                       /* Calculate the window shape                        */
                double window[],        /* Output --- window shape values                    */
                int len,                /* Input --- length of window: long, medium, short   */
                Window_shape wfun_select);      /* Input --- selected window shape       */
```

Noiseless Coding for BSAC

```
void bsac_decode_init();                /* Initial BSAC-decoder settings                     */

int bsac_decode_frame(                  /* Decode bits to MDCT coefficients                  */
                int target_layer                /*  Input – target layer to be decoded */
                BsBitStream *fixed_stream,      /*  Input --- bitstream                   */
                double *spectral_line_vector,   /*  Output spectrum*/
                int *block_type,                /* Output --- block (window) type         */
                Window_shape *window_shape);    /* Output --- window shape                */
```

Encoder functions

Interleaved Vector Quantization and Spectrum Normalization

```
void ntt_tf_coder(                      /* time/frequency mapping coder                      */
                double sig[],           /* input --- time domain signal                      */
                ntt_INDEX  *indexp,     /* output --- index packet                           */
                int    iframe);         /* input --- frame number                            */

void ntt_tf_requantize_spectrum(        /* quantizer for flattened MDCT coefficients         */
                ntt_INDEX *indexp,      /* output --- index packet                           */
                double flat_spectrum[]);/* input --- flattened MDCT coefficients             */

void ntt_enc_lpc_spectrum(              /* quantizer of LPC coefficients                     */
                double sig[],           /* Input --- input time domain signal                */
                int    w_type,          /* Input --- window type                             */
                int    index_lsp[ntt_N_SUP_MAX][ntt_LSP_NIDX_MAX], /* Out:   LSP index*/
                double lpc_spectrum[]); /* Output ---reconstructed reciprocal LPC envelope   */

void ntt_tf_pre_process(                /*  preprocess  before transform                     */
                double sig[],           /* Input --- input time domain signal                */
                ntt_INDEX     *indexp,  /* Output --- index packet                           */
                ntt_PARAM *param_ntt,   /* Output --- parameter packet                       */
```

```
                int   med_sw,           /* input  ---                                   */
                int    InitFlag);       /* input  --- init flag                         */

void ntt_tf_time2freq(                  /* windowing and MDCT                           */
                double sig[],           /* Input --- time domain signal                 */
                int    w_type,          /* Input --- window type                        */
                double spectrum[]);     /* Output --- MDCT coefficients                 */

void ntt_tf_proc_spectrum(              /* Spectral normalization                       */
                double sig[],           /* Input --- time domain signal:                */
                double spectrum[],      /* Input --- MDCT coefficients                  */
                ntt_INDEX  *indexp,     /* In/Out --- index packet                      */
                double lpc_spectrum[],  /* Output --- reconstructed reciprocal LPC spectrum */
                double bark_env[],      /* Output --- reconstructed Bark-scale envelope */
                double pitch_component[],       /* Output --- reconstructed pitch components*/
                double gain[],          /* Output --- reconstructed value of global gain */
                ntt_PARAM param_ntt);   /* In/Out: --- parameter packet                 */

void ntt_freq_domain_pre_process(       /*  preprocess on MDCT                          */
                double spectrum[],      /* Input --- MDCT coefficient                   */
                double lpc_spectrum[],  /* Input --- LPC spectral envelope              */
                ntt_PARAM param_ntt,    /* Input --- prameter packet                    */
                int    w_type,          /* Input --- window type                        */
                double spectrum_out[],  /* Output: --- process ed MDCT coefficients                 */
                double lpc_spectrum_out[]);     /* Outpu --- processed LPC envelope:    */


void ntt_tf_quantize_spectrum(          /* WVQ of normalized MDCT subvectors            */
                double spectrum[],      /* Input --- MDCT coefficient:                  */
                double lpc_spectrum[],  /* Input --- reciprocal LPC spectrum            */
                double bark_env[],      /* Input --- Bark-scale envelope                */
                double pitch_component[],       /* Input -- reconstructed pitch components  */
                double gain[],          /* Input ---  reconstructed value of global gain */
                double perceptual_weight[],     /* input --- weight for distortion measure  */
                ntt_INDEX  *indexp,     /* In/Out --- index packet                      */
                ntt_PARAM *param_ntt);/* In/Out --- parameter packet                    */

void ntt_tf_perceptual_model(           /*  perceptual weight generator                 */
                double lpc_spectrum[],  /* Input --- reciprocal LPC envelope            */
                double bark_env[],      /* Input --- Bark-scal envelope                 */
                double gain[],          /* Input --- reconstructed value of global gain */
                int      w_type,        /* Input --- window type                        */
                double spectrum[],      /* Input --- MDCT coefficients:                 */
                double pitch_sequence[],/* Input --- pitch components                   */
                double perceptual_weight[],     /* Output --- weight for distortion measure  */
                ntt_PARAM *param_ntt);/* In/Out --- parameter packet:                   */

int ntt_BitPack(                        /* pack bits for bitstream                      */
                ntt_INDEX *index,       /* input --- index packet                       */
                BsBitStream *stream,    /* output ---bitstream                          */
                int    InitFlag);       /* input --- init flag                          */
```

to be updated from Tampere document N1298

Filterbank Tool

ANNEX C: MSDL Bit Stream Description

```
class adif_header {
        bit(32) adif_id;
        bit(1) copyright_id_present;
        if ( copyright_id_present ) bit(72) copyright_id;
        bit(1) original_copy;
        uint(1) home;
        bit(1) bitstream_type;
        uint(23) bitrate;
        uint(4) num_program_config_elements;
        repeat(num_program_config_elements + 1) {
                if (bitstream_type == 0) uint(20) adif_buffer_fullness;
                program_config_element ProgramConfigElement;
        }
};

aligned(8) class adts0_sequence {
        while (SYNCWORD) adts0_frame Adts0Frame;
};

aligned(8) class adts0_frame {
        adts0_fixed_header Adts0FixedHeader;
        adts0_variable_header Adts0VariableHeader;
        if (protection_bit==0) uint(16) crc_check;
        raw_data_block RawDataBlock[Adts0Header.number_of_raw_data_blocks_in_frame +1];
};

aligned(8) class adts0_fixed_header : bit(12)=SYNCWORD {
        bit(1) ID;
        uint(2) layer;
        bit(1) protection_absent;
        vlc(sampling_frequency_table) sampling_frequency;
        bit(1) private_bit;
        vlc(channel_configuration_table) channel_configuration;
        bit(1) original_copy;
        uint(1) home;
        vlc(emphasis_table) emphasis;
};

aligned(8) class adts0_variable_header {
        bit(1) copyright_id;
        bit(1) copyright_id_start;
        uint(13) frame_length;
        bit(12) end_of_raw_data;
        bit(8) adts0_buffer_fullness;
        uint(2) number_of_raw_data_blocks_in_frame;
};

map program_config_elements_table () {};
```

```
class raw_data_block {
        repeat { syntactic_element SyntacticElement; } until ( [ID_END] );
};
class single_channel_element is syntactic_element :  bit(3) ID_SCE = 0  {
        bit(4) element_instance_tag;
        individual_channel_stream(false) IndividualChannelStream;
};
class  channel_pair_element  is syntactic_element:  bit(3) ID_CPE = 1  {
        int ws=-1;
        bit(4)element_instance_tag;
        bit(1) common_window;
        if( common_window ) {
                ics_info ICSInfo;
                bit(2) ms_mask_present;
                if ( ms_mask_present == 1 ) bit(1) ms_used[max_sfb];
                ws = ICSInfo.window_sequence;
        }
        individual_channel_stream(common_window, ws) IndividualChannelStream0;
        individual_channel_stream(common_window, ws) IndividualChannelStream1;
};
class ics_info {
        uint(3) window_sequence;
        bit(1) window_shape;
        if ( window_sequence == EIGHT_SHORT_SEQUENCE ) {
                uint(4) max_sfb  ;
                bit(7) ScalefactorGrouping;
        } else {
                uint(6) max_sfb;
                bit(1) predictor_data_present;
                if ( predictor_data_present ) {
                        bit(1) predictor_reset;
                        if ( predictor_reset ) uint(5) predictor_reset_index;
                }
                bit(1) prediction_used[ min(max_sfb,PRED_SFB_MAX) ];
        }
};
class  individual_channel_stream(  bit common_window,
                                        int common_window_sequence ) {
        int window_sequence = common_window_sequence;
        int(8) global_gain;
        if(  !common_window ) {
                ics_info ICSInfo;
                window_sequence = ICSInfo.window_sequence;
                int sfb_cb[number_of_coderbands[ICSInfo.window_sequence] - 1];
                        // this defines an array that is not taken from the
                        // bitstream
                rle(sfb_cb, number_of_coderbands[ICSInfo.window_sequence] - 1);
                        // this function fills the sfb_cb table
                        // from what is read in the bitstream
        } else {
                int sfb_cb[number_of_coderbands[common_window_sequence] - 1];
                rle(sfb_cb, number_of_coderbands[common_window_sequence] - 1);
        }
```

```
        bit (1) pulse_data_presentflag;
        if( pulse_data_presentflag ) pulse_data PulseData;
        bit (1) tns_data_presentflag;
        if( tns_data_presentflag ) tns_data(window_sequence) TNSData;
        bit (1) gain_control_data_presentflag;
        if( gain_control_data_presentflag ) gain_control_data GainControlData;
        scalefactor_data( sfb_cb, number_of_coderbands[window_sequence] - 1, global_gain )
ScaleFactorData;
        spectral_data SpectralData;
};
class scalefactor_data( int sfb_cb[], int sfb_cb_length, int global_gain ) {
        repeat (nonzero(sfb_cb, sfb_cb_length )) {
                vlc(hcod_sf_vlc) hcod_sf;
                                // same as before, reordering is not done in MSDL-S
                                // so we only pull the requested number of values from
                                // the bitstream instead of trying to put them in their
                                // definitive place.
        }
};
map nfilt_bits_for_window_type ( window ) {
        // defined in TNS tool description
}
map  tns_length_bits_for_window_type ( window ) {
        // defined in TNS tool description
}
map  tns_order_bits_for_window_type ( window ) {
        // defined in TNS tool description
}

class tns_data( int window_sequence ) {
        int w, filt, coef_bits;
        for ( w=0 ; w < num_windows[window_sequence]; w++ ) {
                uint(nfilt_bits_for_window_type[w]) n_filt;
                if ( n_filt != 0 )bit (1) coef_res;
                if ( coef_res[w] == 1) coef_bits = 4;
                else coef_bits = 3;
                repeat ( n_filt ) {
                        uint(tns_length_bits_for_window_type[w]) length;
                        uint(tns_order_bits_for_window_type[w]) order;
                        if ( order != 0 ) {
                                bit ( 1 ) direction;
                                bit ( 1 ) coef_compress;
                                uint ( coef_bits  ) coef[order];
                        }
                }
        }
};
class spectral_data {
        repeat (i: max_sd[window_sequence]) {
                repeat (sd_nelems[window_sequence][sfb_cb[i]]) {
                        vlc(hcod_vlc[sfb_cb[i]]) sval;
                                // same as before, no reordering
                }
        }
};

class pulse_data  {
        uint(2) number_pulse;
        uint(6) pulse_start_sfb;
        repeat( numberpulse+1 ){
```

```
                uint(5) pulse_offset;
                uint(4) pulse_amp;
        }
};
class coupling_channel_element is syntactic_element: bit(3) ID_CCE = 2 {
        uint(4) element_instance_tag;
        uint(3) number_of_coupled_elements;
        int num_gain_element_lists = number_of_coupled_elements+1;
        repeat ( number_of_coupled_elements+1 ) {
                uint(5) cc_target;
                if( bitfield(cc_target, 4)) {
                        bit(1) cc_l;
                        bit(1) cc_r;
                        if ( cc_l && cc_r ) num_gain_element_lists += 1;
                }
        }
        bit(1) cc_domain;
        bit(1) gain_element_sign;
        uint(2) gain_element_scale;
        individual_channel_stream(0) channelStream;
        repeat(num_gain_element_lists) {
                bit (1) common_gain_element_present;
                if ( common_gain_element_present ) vlc( hcod_sf_vlc ) hcod_sf;
                else {
                        repeat (nonzero(sfb_cb, sfb_cb_length )) {
                                vlc(hcod_sf_vlc) hcod_sf;
                                        // same as before, reordering is not done in MSDL-S
                                        // so we only pull the requested number of values from
                                        // the bitstream instead of trying to put them in their
                                        // definitive place.
                        }
                }
        }
};
class lfe_channel_element is syntactic_element:bit(3) ID_FXE= 3 {
        uint(4) element_instance_tag;
        individual_channel_stream(0) ChannelStream;
};
class data_stream_element is syntactic_element :bit(3) ID_DSE = 4 {
        bit(4) element_instance_tag;
        uint(4) cnt;
        if ( cnt == 15 ) {
                uint(12) esc_l_cnt;
                cnt += esc_l_cnt;
        }
        repeat ( cnt )uint(8) data_stream_byte[element_instance_tag];
};
class program_config_element : bit(3) ID_PCE = 5 {
        bit(4)element_instance_tag;
        bit(2) profile;
        uint(4) num_front_channel_elements;
        uint(4) num_side_channel_elements;
        uint(4) num_back_channel_elements;
        uint(2) num_lfe_channel_elements;
        uint(3) num_assoc_data_elements;
        uint(4) num_valid_cce_elements;
        bit mono_mixdown_present;
        if ( mono_mixdown_present ) uint(4) mono_mixdown_element_number;
        bit stereo_mixdown_present;
        if( stereo_mixdown_present ) uint(4) stereo_mixdown_element_number;
```

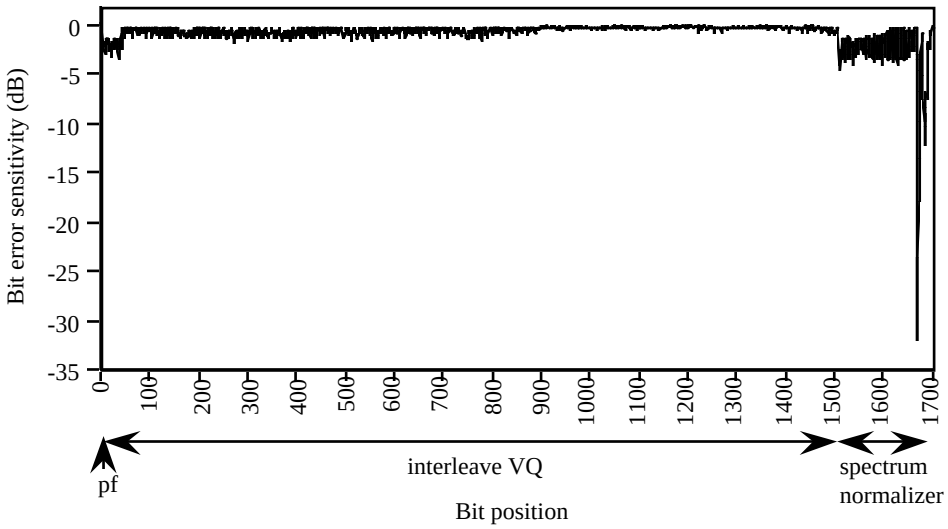
```
        uint(5) front_element_list[num_front_channel_elements];
        uint(5) side_element_list[num_side_channel_elements ];
        uint(5) back_element_list[num_back_channel_elements ];
        uint(4) lfe_element_list[num_lfe_channel_elements ];
        uint(4) assoc_data_element_list[num_assoc_data_elements ];
        uint(4) cce_element_list[num_valid_cce_elements ];
        uint(8) comment_field_bytes;
        uint(8) comment_field_data[comment_field_bytes ];
};
class fill_element is syntactic_element: bit(3) ID_FIL= 6 {
        uint(4) cnt;
        if ( cnt == 15 ) {
                uint(8) esc_cnt;
                cnt += esc_cnt;
        }
        uint(8) fill_byte[cnt];
};
class gain_control_data {
                // nested repeats achieve an arrangement of the data with the inner
                // repeat doing the same as the last index of a C table
                // so in this element, we describe three arrays, adjust_num, alevcode, and aloccode
                // all repeated access fill subsequent elements of the table
                // the inner loop being equivalent to the last index of a C table
        int wd;
        uint(2) maxBand;
        if( window_sequence == ONLY_LONG_SEQUENCE ) {
                repeat( maxBand ){
                        uint(3) adjust_num;
                        repeat( adjust_num ) {
                                uint(4) alevcode;
                                uint(5) aloccode;
                        }}
        } else if ( window_sequence == LONG_START_SEQUENCE ) {
                repeat( maxBand ){
                        for(wd=0;wd<2;wd++) {
                                uint(3) adjust_num;
                                repeat( adjust_num ) {
                                        uint(4) alevcode;
                                        if(wd==0) uint(4) aloccode;
                                        elseuint(2) aloccode;
                                }}}
        } else if ( window_sequence == EIGHT_SHORT_SEQUENCE ) {
                repeat( maxBand ){
                        repeat(8) {
                                uint(3) adjust_num;
                                repeat( adjust_num ) {
                                        uint(4) alevcode;
                                        uint(2) aloccode;
                                }}}
        } else if ( window_sequence == LONG_STOP_SEQUENCE ) {
                repeat( maxBand ){
                        for(wd=0;wd<2;wd++) {
                                uint(3) adjust_num;
                                repeat( adjust_num ) {
                                        uint(4) alevcode;
                                        if(wd==0) uint(4) aloccode;
                                        else uint(5) aloccode;
                                }}}};
```

ANNEX D: Error resilience

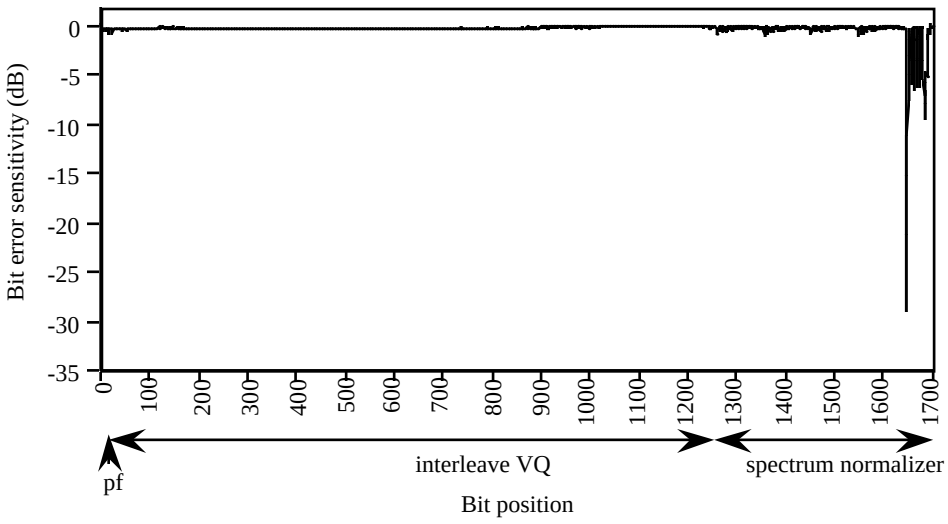
Error resilience for TwinVQ mode

Bit Error Sensitivity

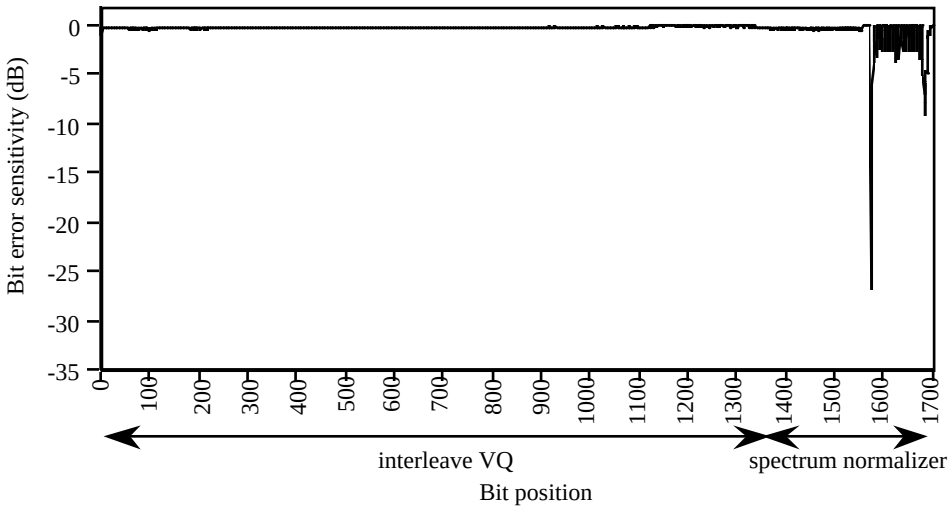
Bit error sensitivity of NTT's T/F coder bits (without window-type bits which are very sensitive to bit error) is shown in Fig.8.1. According to the analysis of error sensitivity, we have found that majority of bits are insensitive to bit error. It is, therefore, feasible to use unequal error protection scheme or Bit Selective Forward Error Control (BS-FEC) scheme to make robust against channel errors by adding small amount of redundancy bits.



(a) LONG



(b) MEDIUM



(c)SHORT

Fig. 8.1. Bit error sensitivity of NTT_VQ mode T/F coder (40 kbit/s).

Suggested Basic Structure of Error Protection

Suggested basic structure of error protection scheme is shown in Fig. 8.2. This is based on he following five major error protection tools to support unequal error protection scheme or BS-FEC (Bit Selective Forward Error Protection) for the bit stream of NTT_VQ mode according to the observation of bit error sensitivities. Tools(1)-(4) are error-detection and error-correction codec for T/F coder's output bitstream; Tool(5) is an error-concealment tool for T/F decoder. These tools can efficiently protect NTT_VQ mode T/F codec even for bad channel conditions.

(1) Error protection tool for window type
(Error correction by block code)
(2) Error protection tool for spectrum normalizer
(Error detection by CRC, correction by convolutional code)
(3) Error protection tool for interleave vector quantizer
(Error detection by parity)
(4) Bit interleave tool
(5) Error protection tool for T/F decoder
(Error concealment by interpolation etc.)

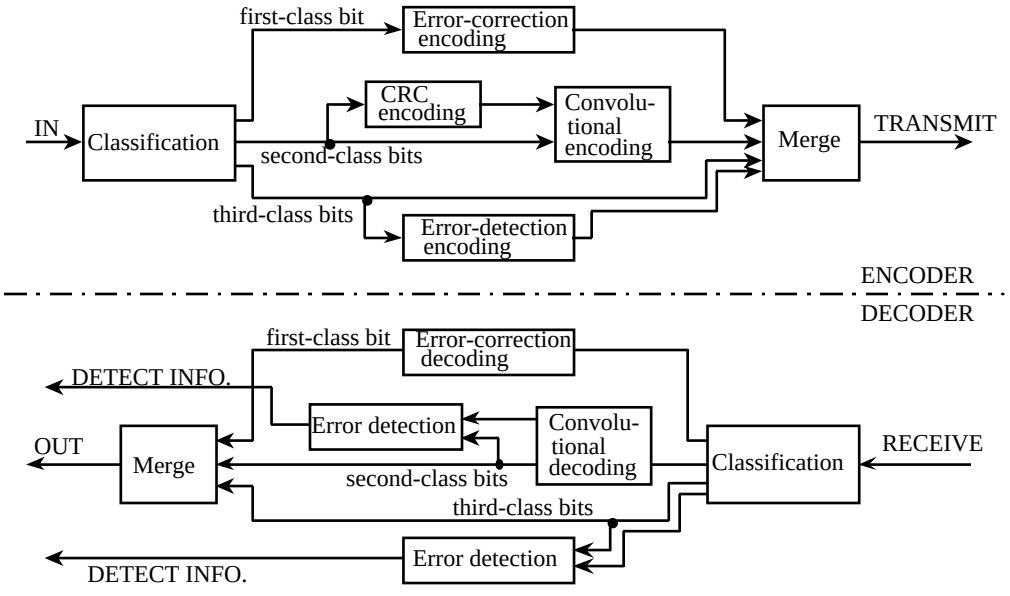


Fig. 8.2. Basic structure of error protection tool.