INTERNATIONAL STANDARD ISO/IEC 23003-3:2012
TECHNICAL CORRIGENDUM 1

Published 2012-09-01

Information technology — MPEG audio technologies —

Part 3: **Unified speech and audio coding**

TECHNICAL CORRIGENDUM 1

Technologies de l'information — Technologies audio MPEG —

Partie 3: Discours unifié et codage audio

RECTIFICATIF TECHNIQUE 1

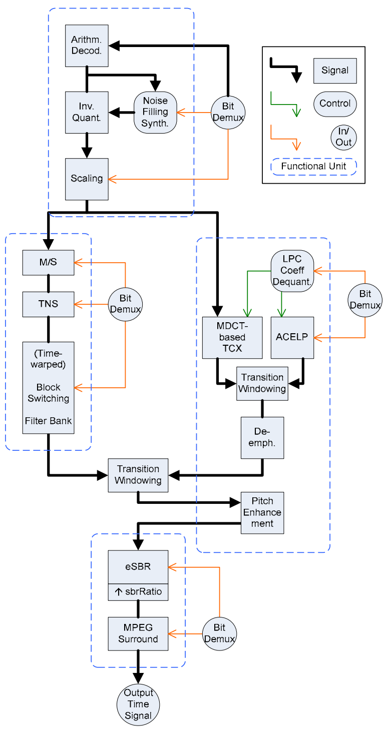
Technical Corrigendum 1 to ISO/IEC 23003-3:2012 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information.*

In 3.2 add at the end:

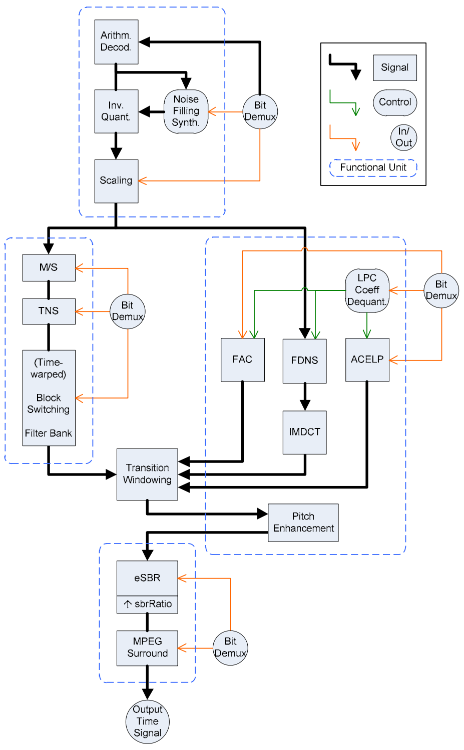
v[] = {a}	This expression indicates that all elements of the array **v** shall be set to the value **a**.

ICS 35.040 **Ref. No. ISO/IEC 23003-3:2012/Cor.1:2012(E)**

In 4.1 replace the diagram:



with:



In 4.2. replace:

The filterbank / block switching tool applies the inverse of the frequency mapping that was carried out in the encoder. An inverse modified discrete cosine transform (IMDCT) is used for the filterbank tool. The IMDCT can be configured to support 120, 128, 240, 256, 480, 512, 960 or 1024 spectral coefficients.

with:

The filterbank / block switching tool applies the inverse of the frequency mapping that was carried out in the encoder. An inverse modified discrete cosine transform (IMDCT) is used for the filterbank tool. The IMDCT can be configured to support 96, 128, 192, 256, 384, 512, 768, or 1024 spectral coefficients.

In 5.2 in Table 6, replace:

```
[…]
        case: ID_USAC_EXT
            UsacExtElementConfig();
            break;
    }
}
```

NOTE: UsacSingleChannelElementConfig(), UsacChannelPairElementConfig(), UsacLfeElement-Config() and UsacExtElementConfig() signaled at position elemIdx refer to the corresponding elements in UsacFrame() at the respective position elemIdx.

with

```
[…]
        case: ID_USAC_EXT
            UsacExtElementConfig();
            break;
        }
    }
}
```

NOTE: UsacSingleChannelElementConfig(), UsacChannelPairElementConfig(), UsacLfeElement-Config() and UsacExtElementConfig() signaled at position elemIdx refer to the corresponding elements in UsacFrame() at the respective position elemIdx.

In 5.3.1 in Table 17 replace:

```
[…]
        case: ID_USAC_EXT
            UsacExtElement(usacIndependencyFlag);
            break;
    }
}
```

with

```
[…]
        case: ID_USAC_EXT
            UsacExtElement(usacIndependencyFlag);
            break;
        }
    }
}
```

In 5.3.1 in Table 21 replace:

```
[…]
        if (usacExtElementUseDefaultLength) {
            usacExtElementPayloadLength = usacExtElementDefaultLength;
        } else {
            usacExtElementPayloadLength = escapedValue(8,16,0);
        }
[…]
```

with:

```
[…]
        if (usacExtElementUseDefaultLength) {
            usacExtElementPayloadLength = usacExtElementDefaultLength;
        } else {
            usacExtElementPayloadLength;                       8        uimsbf
            if (usacExtElementPayloadLength==255) {
                valueAdd                                       16       uimsbf
                usacExtElementPayloadLength += valueAdd - 2;
            }
        }
[…]
```

In 5.3.2 in Table 23 replace:

```
[…]
    if (nrChannels == 2) {
        StereoCoreToolInfo(core_mode);
    }
[…]
```

with

```
[…]
    if (nrChannels == 2) {
        StereoCoreToolInfo(core_mode, indepFlag);
    }
[…]
```

In 5.3.2 in Table 24 replace:

Syntax	No. of bits	Mnemonic
StereoCoreToolInfo(core_mode)		
{		
if (core_mode[0] == 0 && core_mode[1] == 0) {		
tns_active;	**1**	**uimsbf**
common_window;	**1**	**uimsbf**
if (common_window) {		
ics_info();		
common_max_sfb;	**1**	**uimsbf**
if (common_max_sfb == 0) {		

```
                if (window_sequence == EIGHT_SHORT_SEQUENCE) {
                    max_sfb1;                                   4      uimsbf
                } else {
                    max_sfb1;                                   6      uimsbf
                }
            } else {
                max_sfb1 = max_sfb;
            }
            max_sfb_ste = max(max_sfb, max_sfb1);
            ms_mask_present;                                    2      uimsbf
            if ( ms_mask_present == 1 ) {
                for (g = 0; g < num_window_groups; g++) {
                    for (sfb = 0; sfb < max_sfb; sfb++) {
                        ms_used[g][sfb];                        1      uimsbf
                    }
                }
            }
            if (ms_mask_present == 3) {
                cplx_pred_data();
            } else {
                alpha_q_re[g][sfb] = 0;
                alpha_q_im[g][sfb] = 0;
            }
        }
[…]
```

with

```
Syntax                                                          No. of bits  Mnemonic
StereoCoreToolInfo(core_mode, indepFlag)
{
    if (core_mode[0] == 0 && core_mode[1] == 0) {
        tns_active;                                             1            uimsbf
        common_window;                                          1            uimsbf
        if (common_window) {
            ics_info();
            common_max_sfb;                                     1            uimsbf
            if (common_max_sfb == 0) {
                if (window_sequence == EIGHT_SHORT_SEQUENCE) {
                    max_sfb1;                                   4            uimsbf
                } else {
                    max_sfb1;                                   6            uimsbf
                }
            } else {
                max_sfb1 = max_sfb;
            }
            max_sfb_ste = max(max_sfb, max_sfb1);
            ms_mask_present;                                    2            uimsbf
            if ( ms_mask_present == 1 ) {
                for (g = 0; g < num_window_groups; g++) {
                    for (sfb = 0; sfb < max_sfb_ste; sfb++) {
                        ms_used[g][sfb];                        1            uimsbf
                    }
                }
            }
            if (ms_mask_present == 3) {
                cplx_pred_data(max_sfb_ste, indepFlag);
            } else {
                    alpha_q_re[][] = {0};
                    alpha_q_im[][] = {0};
                }
            }
[…]
```

In 5.3.2 in Table 38 – Syntax of arith_data, replace:

pki = arith_get_pk(*c* + *esc_nb<<17)*

with:

pki = arith_get_pk(*c* + *(esc_nb<<17));*

In 6.1.1.1 replace:

usacSamplingFrequency	Output sampling frequency of the decoder coded as unsigned integer value in case usacSamplingFrequencyIndex equals zero.

with:

usacSamplingFrequency	Output sampling frequency of the decoder coded as unsigned integer value in case usacSamplingFrequencyIndex is equal to the escape value.

In 6.1.1.1 add definition of ***bs_pvc*** *by replacing:*

bs_interTes	This flag signals the usage of the inter-TES tool in SBR.

with

bs_interTes	This flag signals the usage of the inter-TES tool in SBR.
bs_pvc	This flag signals the usage of the PVC tool in SBR.

In 6.1.1.2, replace:

bsStereoSbr	This flag signals the usage of the stereo SBR in combination with MPEG Surround decoding.

with

bsStereoSbr	This flag signals the usage of the stereo SBR in combination with MPEG Surround decoding. The value of bsStereoSbr is defined by stereoConfigIndex (see Table 72).

In 6.2.9.2.1 and in the headline of 6.2.9.2.3 replace:

scalefactor_data

with

scale_factor_data

In 6.2.9.2.4 replace

fd_channelffeh_stream()

with

fd_channel_stream()

In 6.2.9.4 replace:

As explain in ISO/IEC 14496-3:2009, 4.5.2.3.4, the width of the scalefactor bands is built in imitation of the critical bands of the human auditory system. For that reason the number of scalefactor bands in a spectrum and their width depend on the transform length and the sampling frequency. Table 4.129 to Table 4.147, in ISO/IEC 14496-3:2009, 4.5.4, list the offset to the beginning of each scalefactor band on the transform lengths 1024 (960) and 128 (120) and on the sampling frequencies.
For a transform length of 768 samples, the scale factor bands at $4/3 \cdot samplingfr\ equency$ are used. In case a shorter transform length (dependent on coreCoderFrameLength) is used, swb_offset_long_window and swb_offset_short_window are limited to the size of the transform length, and num_swb_long_window and num_swb_short_window is determined according to the following pseudo code

with:

As explained in ISO/IEC 14496-3:2009, 4.5.2.3.4, the width of the scalefactor bands is built in imitation of the critical bands of the human auditory system. For that reason the number of scalefactor bands in a spectrum and their width depend on the transform length and the sampling frequency. Table 4.129 to Table 4.147, in ISO/IEC 14496-3:2009, 4.5.4, list the offset to the beginning of each scalefactor band on the transform lengths 1024 and 128 and on the sampling frequencies (window length of 2048 and 256).
For a transform length of 768 samples, the same 1024-based scalefactor band tables are used, but those corresponding to $4/3 \cdot samplingfr\ equency$. In case a shorter transform length (dependent on coreCoderFrameLength) is used, swb_offset_long_window and swb_offset_short_window are limited to the size of the transform length, and num_swb_long_window and num_swb_short_window is determined according to the following pseudo code:

In 6.2.13.2 replace the text block:

bsOttBandsPhase	defines the number of IPD parameter bands. If bsOttBandsPhasePresent==0, …

with:

bsOttBandsPhase	defines the number of MPS parameter bands where phase coding is used. If bsOttBandsPhasePresent==0, …

In 7.4.3 replace the pseudo code:

```
/*Input variables*/
c      /* old state context */
i      /* Index of the 2-tuple to decode in the vector */
N      /* Window Length */
/*Output value*/
c      /*updated state context*/
c = arith_get_context(c,i,N) {
       c = c>>4;
       if (i<N/4-1)
              c = c + (q[0][i+1]<<12);
       c = (c&0xFFF0);
       if (i>0)
              c = c + (q[1][i-1]);
       if (i > 3) {
              if ((q[1][i-3] + q[1][i-2] + q[1][i-1]) < 5)
                     return(c+0x10000);
       }
       return (c);
}
```

with:

```
/*Input variables*/
c      /* old state context */
i      /* Index of the 2-tuple to decode in the vector */
N      /* Window Length */
/*Output value*/
c      /*updated state context*/
c = arith_get_context(c,i,N) {
       c = (c & 0xFFFF)>>4;
       if (i<N/4-1)
              c = c + (q[0][i+1]<<12);
       c = (c&0xFFF0);
       if (i>0)
              c = c + (q[1][i-1]);
       if (i > 3) {
              if ((q[1][i-3] + q[1][i-2] + q[1][i-1]) < 5)
                     return(c+0x10000);
       }
       return (c);
}
```

Further in 7.4.3 replace the following pseudo code:

```
/*input variables*/
offset       /* number of decoded 2-tuples */
N            /* Window length */
x_ac_dec     /* vector of decoded spectal coefficients */
arith_finish(x_ace_dec,offset,N)
{
       for (i=offset ;i<N/4;i++) {
              x_ac_dec[2*i] = 0;
              x_ac_dec[2*i+1] = 0;
              q[1][i] = 1;
       }
}
```

with:

```
/*helper function*/
void arith_rewind_bitstream(offset);
       /* move the bitstream position indicator backward by 'offset' bits*/

/*input variables*/
offset       /* number of decoded 2-tuples */
N            /* Window length */
x_ac_dec     /* vector of decoded spectal coefficients */
```

```
arith_finish(x_ace_dec,offset,N)
{
       arith_rewind_bitstream(14);

       for (i=offset ;i<N/4;i++) {
              x_ac_dec[2*i] = 0;
              x_ac_dec[2*i+1] = 0;
              q[1][i] = 1;
       }
}
```

In 7.4.3 in function `arith_decode()`, *replace:*

```
value = (val<<1)…
```

with:

```
value = (value<<1)…
```

Further in 7.4.3, replace:

```
high = low +(range*cum_freq[symbol-1])>>14 - 1
```

with:

```
high = low +((range*cum_freq[symbol-1])>>14) - 1;
```

In 7.4.3 replace:

…with the value *c&esc_nb<<17* as input argument,…

with

…with the value *c* + *(esc_nb<<17)* as input argument,…

In 7.4.3 replace:

…the function *get_pk()*…

with:

…the function *arith_get_pk()*…

Also in 7.4.3 replace:

…If the condition *(esc_nb>0 && m==0)* is true …

with:

…If the condition *(m==0 && lev>0)* is true,…

Further down in the same subclause 7.4.3, replace:

```
arith_finish(x_ace_dec,offset,N)
```

with:

```
arith_finish(x_ac_dec,offset,N)
```

In 7.5.1 replace:

The general description of the SBR tool can be found in ISO/IEC 14496-3:2009, 4.6.18.
The above mentioned SBR tool shall be modified as described below.

with:

The general description of the SBR tool can be found in ISO/IEC 14496-3:2009, 4.6.18.
The complex-exponential phase-shifting is outlined in ISO/IEC 14496-3:2009, 4.6.18.4.4. In USAC it shall be fixed to the default standard operation as defined in 4.6.18.4.1.
The above mentioned SBR tool shall be modified as described below.

In 7.5.1.1 add the following line:

numTimeSlots	number of SBR envelope time slots; is always 16.

In 7.5.1.4 replace:

Within one SBR frame there can be either one or two noise floors. The noise floor time borders are derived from the SBR envelope time border vector according to: …

with:

Independent of bs_pvc_mode within one SBR frame there can be either one or two noise floors.
If bs_pvc_mode is zero, the noise floor time borders are derived from the SBR envelope time border vector according to: …

In 7.5.1.5.2 replace the following text:

If bs_pvc_mode in not zero, the SBR envelope time border vector of the current SBR frame, $\mathbf{t}_E$ is calculated according to:

$$\mathbf{t}_E = \begin{cases} \left[\mathbf{bs\_var\_len'},\ numTimeSlots + \mathbf{bs\_var\_len}\right] & ,bs\_num\_env = 1 \\ \left[\mathbf{bs\_var\_len'},\ \mathbf{bs\_noise\_position},\ numTimeSlots + \mathbf{bs\_var\_len}\right] & ,bs\_num\_env = 2 \end{cases}$$

where

$\mathbf{bs\_var\_len}$ ' is $\mathbf{bs\_var\_len}$ of the previous SBR frame.

$$bs\_num\_env = \begin{cases} 1 & \text{if bs\_noise\_position} = 0 \\ 2 & \text{otherwise} \end{cases}$$

with:

If bs_pvc_mode is not zero, the SBR envelope time border vector of the current SBR frame, $\mathbf{t}_E$ is calculated according to:

$$L_E = \begin{cases} 1 & \text{if bs\_noise\_position} = 0 \\ 2 & \text{otherwise} \end{cases}$$

$$\mathbf{t}_E = \begin{cases} \left[\mathbf{var\_len'},\ numTimeSlots + \mathbf{bs\_var\_len}\right] & , L_\mathrm{E} = 1 \\ \left[\mathbf{var\_len'},\ \mathbf{bs\_noise\_position},\ numTimeSlots + \mathbf{bs\_var\_len}\right] & , L_\mathrm{E} = 2 \end{cases}$$

where

$\mathbf{var\_len'} = \mathbf{t'}_E\,[L'_E] - numTimeSlots$ and $\mathbf{t'}_E$ is the time border vector $\mathbf{t}_E$ of the previous SBR frame and L'_E is the number of envelopes of the previous frame respectively. Note that if bs_pvc_mode'==1 (PVC active in previous frame), it follows that $\mathbf{var\_len'}$ is $\mathbf{bs\_var\_len}$ of the previous SBR frame.

In the same subclause 7.5.1.5.2 replace:

If bs_pvc_mode is not zero, the PVC SBR envelope time border vector of the current SBR frame, $\mathbf{t}_{EPVC}$, is calculated according to:

$$\mathbf{t}_{EPVC} = \begin{cases} \left[t_{first},\ numTimeSlots\right] & , bs\_num\_env = 1 \\ \left[t_{first},\ \text{bs\_noise\_position},\ numTimeSlots\right] & , bs\_num\_env = 2 \end{cases}$$

where

$$t_{first} = \begin{cases} bs\_var\_bord\_1' & , bs\_pvc\_mode' = 0 \text{ and } bs\_pvc\_mode \neq 0 \\ 0 & , \text{otherwise} \end{cases}$$

with:

If bs_pvc_mode is not zero, the PVC SBR envelope time border vector of the current SBR frame, $\mathbf{t}_{EPVC}$, is calculated according to:

$$\mathbf{t}_{EPVC} = \begin{cases} \left[t_{first},\ numTimeSlots\right] & , L_\mathrm{E} = 1 \\ \left[t_{first},\ \text{bs\_noise\_position},\ numTimeSlots\right] & , L_\mathrm{E} = 2 \end{cases}$$

where

$$t_{first} = \begin{cases} \mathbf{var\_len'} & , bs\_pvc\_mode' = 0 \text{ and } bs\_pvc\_mode \neq 0 \\ 0 & , \text{otherwise} \end{cases}$$

and $\mathbf{var\_len'} = \mathbf{t'}_E\,[L'_E] - numTimeSlots$ and $\mathbf{t'}_E$ is the time border vector $\mathbf{t}_E$ of the previous SBR frame and L'_E is the number of envelopes of the previous frame respectively.

In the same subclause 7.5.1.5.2 replace:

If bs_pvc_mode is not zero, the noise floor time borders vectors of the current SBR frame, $\mathbf{t}_Q$ is calculated according to:

$$\mathbf{t}_Q = \begin{cases} \left[\mathbf{t}_E(0), \mathbf{t}_E(1)\right] & , bs\_num\_noise = 1 \\ \left[\mathbf{t}_E(0), \mathbf{t}_E(1), \mathbf{t}_E(2)\right] & , bs\_num\_noise = 2 \end{cases}$$

where

$$bs\_num\_noise = \begin{cases} 1 & \text{if } bs\_noise\_position = 0 \\ 2 & \text{otherwise} \end{cases}$$

with:

If bs_pvc_mode is not zero, the noise floor time borders vectors of the current SBR frame, $\mathbf{t}_Q$ is calculated according to:

$$L_Q = L_E$$

$$\mathbf{t}_Q = \begin{cases} [\mathbf{t}_E(0), \mathbf{t}_E(1)] & , L_Q = 1 \\ [\mathbf{t}_E(0), \mathbf{t}_E(1), \mathbf{t}_E(2)] & , L_Q = 2 \end{cases}$$

In 7.5.1.5.2 on page 96 replace equations as follows:

else, bs_pvc_mode is not zero,

$$\mathbf{S}_{Mapped}(m-k_x,t) = \delta_s(i,t), l_i \le m < u_i, \begin{cases} u_i = \mathbf{F}(i+1, \mathbf{r}(l)) \\ l_i = \mathbf{F}(i, \mathbf{r}(l)) \end{cases}$$

for $0 \le i < \mathbf{n}(\mathbf{r}(l)), \mathbf{t}_E(l) \le t < \mathbf{t}_E(l+1), 0 \le l < L_E$
where

$$\delta_s(i,t) = \begin{cases} 1 & ,1 \in \{\mathbf{S}_{IndexMapped}(j-k_x,t) : \mathbf{F}(i,\mathbf{r}(l)) \le j < \mathbf{F}(i+1,\mathbf{r}(l)), \mathbf{t}_E(l) \le t < \mathbf{t}_E(l+1), 0 \le l < L_E\} \\ 0 & ,\text{otherwise} \end{cases}$$

with:

else, bs_pvc_mode is not zero,

$$\mathbf{S}_{Mapped}(m-k_x,t) = \delta_s(i,t), l_i \le m < u_i, \begin{cases} u_i = \mathbf{F}(i+1, \mathbf{r}(l)) \\ l_i = \mathbf{F}(i, \mathbf{r}(l)) \end{cases}$$

for $0 \le i < \mathbf{n}(\mathbf{r}(l)), \mathbf{t}_{EPVC}(l) \le t < \mathbf{t}_{EPVC}(l+1), 0 \le l < L_E$
where

$$\delta_s(i,t) = \begin{cases} 1 & ,1 \in \{\mathbf{S}_{IndexMapped}(j-k_x,t) : \mathbf{F}(i,\mathbf{r}(l)) \le j < \mathbf{F}(i+1,\mathbf{r}(l)), \mathbf{t}_{EPVC}(l) \le t < \mathbf{t}_{EPVC}(l+1), 0 \le l < L_E\} \\ 0 & ,\text{otherwise} \end{cases}$$

In 7.5.1.5.2 replace the equation of $\mathbf{Q}_{Mapped}$ in case of pvc_mode is not zero as follows.Replace:

$$\mathbf{Q}_{Mapped}(m-k_x,t) = \begin{cases} \mathbf{Q'}_{PreMapped}(m-k_x, t+numTimeSlots) & , 0 \le t < \mathbf{t'}_\mathrm{E}(L'_E) - numTimeSlots \\ \mathbf{Q}_{PreMapped}(m-k_x,t) & , \mathbf{t}_\mathrm{E}(0) \le t < \mathbf{t}_\mathrm{E}(L_E) \end{cases}$$

with:

$$\mathbf{Q}_{Mapped}(m-k_x,t) = \begin{cases} \mathbf{Q'}_{PreMapped}(m-k_x, t+numTimeSlots) & , 0 \le t < \mathbf{t}_\mathrm{E}(0) \\ \mathbf{Q}_{PreMapped}(m-k_x,t) & , \mathbf{t}_\mathrm{E}(0) \le t < \mathbf{t}_\mathrm{E}(L_E) \end{cases}$$

In 7.5.1.5.2 replace the equation for $\mathbf{S}_{IndexMapped}$ *in case of pvc_mode is not zero as follows. Replace:*

$$\mathbf{S}_{IndexMapped}(m-k_x,t)=\begin{cases}\mathbf{S'}_{IndexPreMapped}(m-k_x,t+numTimeSlots), & 0\le t<\mathbf{t'}_E(L'_E)-numTimeSlots\\ \mathbf{S}_{IndexPreMapped}(m-k_x,t) & ,\mathbf{t}_E(0)\le t<\mathbf{t}_E(L_E)\end{cases}$$

with:

$$\mathbf{S}_{IndexMapped}(m-k_x,t)=\begin{cases}\mathbf{S'}_{IndexPreMapped}(m-k_x,t+numTimeSlots), & 0\le t<\mathbf{t}_E(0)\\ \mathbf{S}_{IndexPreMapped}(m-k_x,t) & ,\mathbf{t}_E(0)\le t<\mathbf{t}_E(L_E)\end{cases}$$

In 7.5.1.5.4 replace equations as follows:

else, *bs_pvc_mode* is not zero,

$$\mathbf{Q}_M(m,t)=\sqrt{\mathbf{E}_{OrigMapped}(m,t)\cdot\frac{\mathbf{Q}_{Mapped}(m,t)}{1+\mathbf{Q}_{Mapped}(m,t)}}\ ,0\le m<M,\mathbf{t}_E(l)\le t<\mathbf{t}_E(l+1),0\le l<L_E$$

with:

else, *bs_pvc_mode* is not zero,

$$\begin{aligned}\mathbf{Q}_M(m,t)=&\sqrt{\mathbf{E}_{OrigMapped}(m,t)\cdot\frac{\mathbf{Q}_{Mapped}(m,t)}{1+\mathbf{Q}_{Mapped}(m,t)}}\ ,\\ &0\le m<M,\\ &\mathbf{t}_{EPVC}(l)\le t<\mathbf{t}_{EPVC}(l+1),\\ &0\le l<L_E\end{aligned}$$

Further, replace:

else, *bs_pvc_mode* is not zero,

$$\mathbf{S}_M(m,t)=\sqrt{\mathbf{E}_{OrigMapped}(m,t)\cdot\frac{\mathbf{S}_{IndexMapped}(m,t)}{1+\mathbf{Q}_{Mapped}(m,t)}}\ ,0\le m<M,\mathbf{t}_E(l)\le t<\mathbf{t}_E(l+1),0\le l<L_E$$

with:

else, *bs_pvc_mode* is not zero,

$$\begin{aligned}\mathbf{S}_M(m,t)=&\sqrt{\mathbf{E}_{OrigMapped}(m,t)\cdot\frac{\mathbf{S}_{IndexMapped}(m,t)}{1+\mathbf{Q}_{Mapped}(m,t)}}\ ,\\ &0\le m<M,\\ &\mathbf{t}_{EPVC}(l)\le t<\mathbf{t}_{EPVC}(l+1),\\ &0\le l<L_E\end{aligned}$$

In 7.5.2.2 replace:

$$polyfit\left(3, k_0, x\_lowband, lowEnv, lowEnvSlope\right);$$

with:

$$\mathrm{polyfit}\left(3, k_0, x\_lowband, lowEnv, polyCoeffs\right);$$

$$lowEnvSlope(k) = \sum_{i=0}^{3} polyCoeffs(3-i) \cdot x\_lowband(k)^i$$

In 7.5.5.2 replace:

$$lowEnv\left(k\right) = 10\log_{10}\left(\frac{\phi_k\left(0,0\right)}{numTimeSlots \cdot RATE + 6}\right) \quad ,0 \le k < k_0$$

with:

$$lowEnv\left(k\right) = 10\log_{10}\left(\frac{\phi_k\left(0,0\right)}{(numTimeSlots + 3) \cdot RATE}\right) \quad ,0 \le k < k_0$$

In 7.5.6.3 replace

$$E(ib,t) = \frac{\sum_{i=t_{HFGen}}^{RATE-1+t_{HFGen}} X_{low}(ib, RATE \cdot t + i) \cdot X^{*}{}_{low}(ib, RATE \cdot t + i)}{RATE}$$

with

$$E(ib,t) = \frac{\sum_{i=t_{HFAdj}}^{RATE-1+t_{HFAdj}} X_{low}(ib, RATE \cdot t + i) \cdot X^{*}{}_{low}(ib, RATE \cdot t + i)}{RATE}$$

In 7.5.6.5 replace:

$$\hat{E}(k, t + \frac{t_{HFGen} - t_{HFAdj}}{RATE}) = 10^{\frac{\hat{E}sg(ksg,t)}{10}}$$

with

$$\hat{E}(k,t) = 10^{\frac{\hat{E}sg(ksg,t)}{10}}$$

In 7.9.3.2 replace:

Depending on the **window_sequence** and **window_shape** element different transform windows are used. A combination of the window halves described as follows offers all possible window_sequences. Window lengths specified below are dependent on the core-coder frame length. Numbers are listed for coreCoderFrameLength of 1024 (960, 768).

with:

Depending on the **window_sequence** and **window_shape** element different transform windows are used. A combination of the window halves described as follows offers all possible window_sequences. Window lengths specified below are dependent on the core-coder frame length. Numbers are listed for coreCoderFrameLength of 1024 (768).

Further below in 7.9.3.2 replace:

α = kernel window alpha factor, α = $\begin{cases} 4 \text{ for N =2048 (1920, 1536)} \\ 6 \text{ for N =256 (240, 192)} \end{cases}$

with:

α = kernel window alpha factor, $\alpha = \begin{cases} 4 \text{ for } N = 2048 \ \ (1536) \\ 6 \text{ for } N = 256 \ \ (192) \end{cases}$

In all of 7.9.3.2 remove the mentioning of 1920 and 240 sample window lengths.

In the rest of the document remove all further references to 960/120 based frame length or 1920/240 based window length and add reference to the 768/96 transform length (1536/192 window length) coding if appropriate and if not already present. Do so also in formulas, equations and figures.

In 7.11.1, add to the end:

Unlike the delay introduced by MPEG Surround decoder as defined in ISO/IEC 23003-1:2007, 4.5, only High Quality decoding is supported in MPS212. It is noted that this implies that the delay of 5 QMF samples prior to the Nyquist analysis filterbanks shall not be inserted.

In 7.11.2.3.4, replace:

If **bsPhaseCoding** == 1 and **bsResidualCoding** == 1, the matrix $R_2^{l,m}$ is defined as following:

with

If **bsResidualCoding** == 1, the matrix $R_2^{l,m}$ is defined as follows (where, if **bsPhaseCoding** == 1, the transmitted $IPD^{l,m}$ values are used, and where, if **bsPhaseCoding** == 0, the value $IPD = 0$ is used for all parameter sets l and processing bands m):

In 7.11.2.3.4, add a sentence after the equation to calculate $CLD_{lin}^{l,m}$ *as follows:*

using

$$CLD_{lin}^{l,m} = 10^{\frac{CLD^{l,m}}{10}}$$

It is noted that resBands refers to the value of **bsResidualBands**, i.e. the number of MPS parameter bands where residual coding is used.

In 7.11.2.5 replace:

For the 2-1-2 configuration, the frequency axis is divided into four different regions according to *bsDecorrConfig*~~-~~= 0 and only one decorrelator is used, *X* = *0*.

with:

For the 2-1-2 configuration, the frequency axis is divided into up to four different regions according to *bsDecorrConfig* but only one decorrelator is used, *X = 0*.

At the end of 7.13 add a new subclause:

7.13.12 LPC initialization at decoder start-up

In frames where the first decoded frame is LPD and the initial filter LPC0 is not transmitted within the bitstream, the LPD core decoder is reset as for a regular start-up. In particular, the ACELP decoder is initialized as described in 7.14.3. Additionally, the LSF vector corresponding to the LPC filter LPC0 is set to the value specified in Table COR1.2 before inverse LPC quantization.

Right after inverse LPC quantization, the LSF vector corresponding to LPC0 is reset as follows:

$$LSF_0 = \frac{\text{mean\_lsf} + LSF_i}{2},$$

where mean_lsf is the mean LSF vector specified in Table COR1.2 and LSF_i is the LSF vector corresponding to the LPC filter of frame *i*, *i* being determined as follows:

Table COR1.1 — Value of i for calculating LSF_0

Condition	Value of i
mod[0]<2	1
mod[0]=2	2
mod[0]=3	4

This operation corresponds to setting the LSF vector corresponding to LPC0 to average between the mean LSF vector and the nearest decoded LSF vector (which depends on the coding mode).

Table COR1.2 — Mean LSF vector for initialization

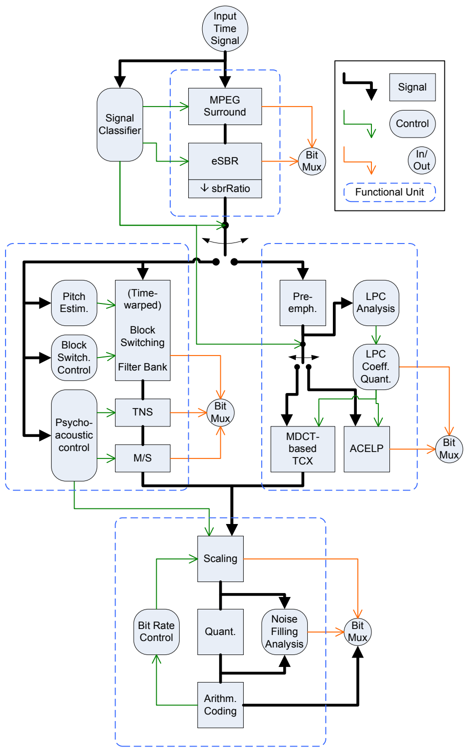
j	mean_lsf(j)
1	394,21
2	754,45
3	1209,89
4	1580,47
5	1953,97
6	2325,80
7	2684,41
8	3038,39
9	3392,56
10	3744,71
11	4118,14
12	4483,09
13	4862,21
14	5219,69
15	5594,41
16	5945,73

Amend 7.16.3, list item 4, as follows:

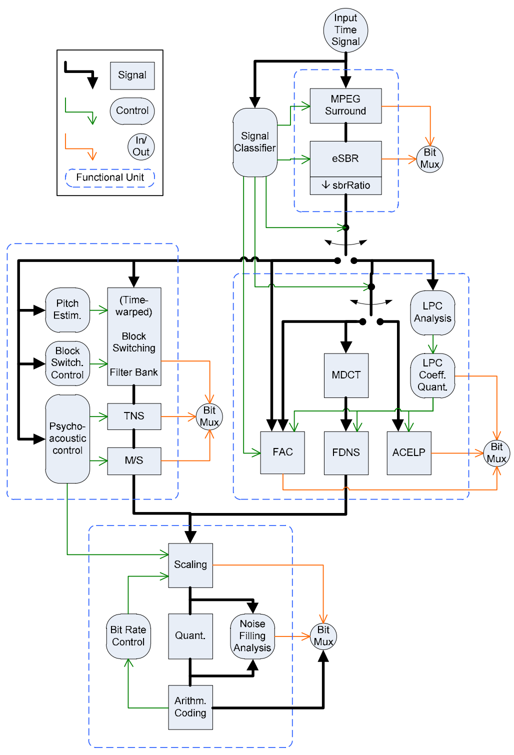
4. Compute the inverse DCT-IV to the gain-scaled FAC data to obtain the equivalent time-domain samples.
   - The FAC transform length, fac_length, is by default equal to coreCoderFrameLength/8
   - For transitions with short blocks, this length is reduced to coreCoderFrameLength/16

   In the case of transitions between ACELP and FD mode, a multiplicative factor of (2/fac_length) is applied to the output of the inverse DCT-IV.

In Annex B.1 replace diagram:



with:



In Annex B.16.3, Figure B.8 replace:

Quantized LSFs

with:

Quantized weighted residual LSFs

In Annex B.21 replace:

- mpegsMuxMode = 2

with

- bsResidualCoding = 1

Furthermore, throughout the whole document replace “ari_” *with* “arith_”