Technical Specification Group Services and System Aspects ***TSGS#19(03)0090***
Meeting #19, Birmingham, UK, 17 - 20 March 2003

Source: **TSG-SA WG4**

Title: CRs to TS 26.204 - Corrections (Release 5)

Document for: **Approval**

Agenda Item: **7.4.3**

The following CRs, agreed at the TSG-SA WG4 meeting #25/#25bis, are presented to TSG SA #19 for approval.

Spec	CR	Rev	Phase	Subject	Cat	Vers	WG	Meeting	S4 doc
26.204	001	1	Rel-5	Correction to log(0) error in VAD decision with low SNR input signals	F	5.0.0	S4	TSG-SA WG4#25	S4-030085
26.204	002	1	Rel-5	Correction to decoder with input of long sequence of NO_DATA frames	F	5.0.0	S4	TSG-SA WG4#25	S4-030084
26.204	003	1	Rel-5	Correction to "D_UTIL_pow2" function to be bitexact with TS26.173 counterpart	F	5.0.0	S4	TSG-SA WG4#25	S4-030087
26.204	004	1	Rel-5	MMS compatible i/o format option	F	5.0.0	S4	TSG-SA WG4#25	S4-030088
26.204	005		Rel-5	Correction for handling of RX_NO_DATA frames	F	5.0.0	S4	TSG-SA WG4#25bis	S4-030144
26.204	006	1	Rel-5	Ambiguous expressions in the AMR-WB Floating-point C-Code	F	5.0.0	S4	TSG-SA WG4#25bis	S4-030174

3GPP TSG-SA4 Meeting #25
San Francisco, USA, 20 - 24 January 2003

Tdoc ⌘S4-030085

CR-Form-v7

CHANGE REQUEST

⌘	**26.204**	**CR**	**001**	⌘**rev**	**1**	⌘	Current version:	**5.0.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ ☐ ME ☒ Radio Access Network ☐ Core Network ☒

Title:	⌘	Correction to log(0) error in VAD decision with low SNR input signals		
Source:	⌘	TSG SA WG4		
Work item code:	⌘	AMRWB-FP	***Date:*** ⌘	18/03/2003
Category:	⌘	**F**	***Release:*** ⌘	Rel-5

Use one of the following categories:
F *(correction)*
A *(corresponds to a correction in an earlier release)*
B *(addition of feature),*
C *(functional modification of feature)*
D *(editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:

2	*(GSM Phase 2)*
R96	*(Release 1996)*
R97	*(Release 1997)*
R98	*(Release 1998)*
R99	*(Release 1999)*
Rel-4	*(Release 4)*
Rel-5	*(Release 5)*
Rel-6	*(Release 6)*

Reason for change:	⌘	Logarithm is taken from data having zero or negative value in VAD decision function in cases when low SNR input signal is fed into coder.
Summary of change:	⌘	Input data into logarithmic function is changed to saturate into small positive number.
Consequences if not approved:	⌘	Execution error can occure with low SNR input signal.

Clauses affected:	⌘	enc_dtx.c			
		Y	**N**		
Other specs affected:	⌘		**X**	Other core specifications	⌘
			X	Test specifications	
			X	O&M Specifications	
Other comments:	⌘				

1. enc_dtx.c

Lines 1238 to 1256 before change:

```
/*
 * if SNR is lower than a threshold (MIN_SPEECH_SNR),
 * and increase speech_level
 */
temp = noise_level * MIN_SPEECH_SNR * 8;

if (st->mem_speech_level < temp)
{
   st->mem_speech_level = temp;
}

ilog2_noise_level = (Float32)(-1024.0F * log10(noise_level / 2147483648.0F) / log10(2.0F));

/*
 * If SNR is very poor, speech_level is probably corrupted by noise level. This
 * is correctred by subtracting -MIN_SPEECH_SNR*noise_level from speech level
 */
ilog2_speech_level = (Float32)(-1024.0F * log10((st->mem_speech_level - temp) / 2147483648.0F) / log10(2.0F));
```

after the change:

```
/*
 * if SNR is lower than a threshold (MIN_SPEECH_SNR),
 * and increase speech_level
 */
temp = noise_level * MIN_SPEECH_SNR * 8;

if (st->mem_speech_level <= temp)
{
   st->mem_speech_level = temp;

   /* avoid log10 error */
   temp -= 1E-8F;
}

ilog2_noise_level = (Float32)(-1024.0F * log10(noise_level / 2147483648.0F) / log10(2.0F));

/*
 * If SNR is very poor, speech_level is probably corrupted by noise level. This
 * is correctred by subtracting -MIN_SPEECH_SNR*noise_level from speech level
 */
ilog2_speech_level = (Float32)(-1024.0F * log10((st->mem_speech_level - temp) / 2147483648.0F) / log10(2.0F));
```

3GPP TSG-SA4 Meeting #25
San Francisco, USA, 20 - 24 January 2003

Tdoc ⌘S4-030084

CR-Form-v7

CHANGE REQUEST

⌘	**26.204**	**CR**	**002**	⌘**rev**	**1**	⌘ Current version:	**5.0.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ ☐ ME ☒ Radio Access Network ☐ Core Network ☒

Title:	⌘	Correction to decoder with input of long sequence of NO_DATA frames		
Source:	⌘	TSG SA WG4		
Work item code:	⌘	AMRWB-FP	***Date:*** ⌘	18/03/2003
Category:	⌘	**F** *Use one of the following categories:* ***F*** *(correction)* ***A*** *(corresponds to a correction in an earlier release)* ***B*** *(addition of feature),* ***C*** *(functional modification of feature)* ***D*** *(editorial modification)* Detailed explanations of the above categories can be found in 3GPP TR 21.900.	***Release:*** ⌘	Rel-5 *Use one of the following releases:* *2 (GSM Phase 2)* *R96 (Release 1996)* *R97 (Release 1997)* *R98 (Release 1998)* *R99 (Release 1999)* *Rel-4 (Release 4)* *Rel-5 (Release 5)* *Rel-6 (Release 6)*

Reason for change:	⌘	Counter for time that has ellapsed since last confort noise update overflows. This causes decoder to crash if there is no data input for 256 frames. Frame energy for confort noise can underflow if there is no data to receive for long time, this results maximum energy for confort noise.
Summary of change:	⌘	Confort noise algorithm is changed to withstand longer data interruptions. Counter for since last SID is changed from 8-bit to 16-bit variable to allow the use of saturation function. Saturation is added to the confort noise frame energy variable.
Consequences if not approved:	⌘	Possible “division by zero” error. Decoder is not bit-exact with TS26.173 decoder.

Clauses affected:	⌘	Source code files: • dec_dtx.c • dec_dtx.h 26.204-500.doc: • Table 6: Speech decoder static variables				
		Y	N			
Other specs affected:	⌘		X	Other core specifications	⌘	
			X	Test specifications		
			X	O&M Specifications		
Other comments:	⌘					

1. dec_dtx.c

Lines 205 to 207 before change:

```
    /* evaluate if noise parameters are too old                    */
    /* since_last_sid is reset when CN parameters have been updated */
    st->mem_since_last_sid++;
```

after the change:

```
    /* evaluate if noise parameters are too old                    */
    /* since_last_sid is reset when CN parameters have been updated */
    st->mem_since_last_sid = D_UTIL_saturate(st->mem_since_last_sid + 1);
```

Lines 665 to 670 before change:

```
    st->mem_true_sid_period_inv = (Word16)(0x7FFFFFFF / tmp_int_length);
    st->mem_since_last_sid = 0;
    st->mem_log_en_prev = st->mem_log_en;

    /* subtract 1/8 in Q9 (energy), i.e -3/8 dB */
    st->mem_log_en = (Word16)(st->mem_log_en - 64);
```

after the change:

```
    st->mem_true_sid_period_inv = D_UTIL_saturate((0x02000000 / (tmp_int_length << 10)));
    st->mem_since_last_sid = 0;
    st->mem_log_en_prev = st->mem_log_en;

    /* subtract 1/8 in Q9 (energy), i.e -3/8 dB */
    st->mem_log_en = D_UTIL_saturate(st->mem_log_en - 64);
```

2. dec_dtx.h

Lines 17 to 39 before change:

```
    typedef struct {
      Word16 mem_isf_buf[M * D_DTX_HIST_SIZE];  /* ISF vector history (8 frames)*/
      Word16 mem_isf[M];            /* ISF vector                        */
      Word16 mem_isf_prev[M];       /* Previous ISF vector               */
      Word16 mem_log_en_buf[D_DTX_HIST_SIZE];/* logarithmic frame energy history*/
      Word16 mem_true_sid_period_inv;  /* inverse of true SID update rate      */
      Word16 mem_log_en;            /* logarithmic frame energy          */
      Word16 mem_log_en_prev;       /* previous logarithmic frame energy      */
      Word16 mem_cng_seed;          /* Comfort noise excitation seed          */
      Word16 mem_hist_ptr;          /* index to beginning of LSF history       */
      Word16 mem_dither_seed;       /* comfort noise dithering seed           */
      Word16 mem_cn_dith;           /* background noise stationarity information*/

      UWord8 mem_dec_ana_elapsed_count;/* counts elapsed speech frames after DTX*/
      UWord8 mem_dtx_global_state;  /* DTX state flags                    */
      UWord8 mem_since_last_sid;    /* number of frames since last SID frame    */
      UWord8 mem_data_updated;      /* flags CNI updates                   */
      UWord8 mem_dtx_hangover_count;/* counts down in hangover period           */
      UWord8 mem_sid_frame;         /* flags SID frames                    */
      UWord8 mem_valid_data;        /* flags SID frames containing valid data   */
```

```
     UWord8 mem_dtx_hangover_added;/* flags hangover period at end of speech   */

   } D_DTX_State;
```

after the change:

```
    typedef struct {
      Word16 mem_isf_buf[M * D_DTX_HIST_SIZE];  /* ISF vector history (8 frames)*/
      Word16 mem_isf[M];            /* ISF vector                              */
      Word16 mem_isf_prev[M];       /* Previous ISF vector                     */
      Word16 mem_log_en_buf[D_DTX_HIST_SIZE];/* logarithmic frame energy history*/
      Word16 mem_true_sid_period_inv;  /* inverse of true SID update rate       */
      Word16 mem_log_en;            /* logarithmic frame energy                */
      Word16 mem_log_en_prev;       /* previous logarithmic frame energy       */
      Word16 mem_cng_seed;          /* Comfort noise excitation seed           */
      Word16 mem_hist_ptr;          /* index to beginning of LSF history       */
      Word16 mem_dither_seed;       /* comfort noise dithering seed            */
      Word16 mem_cn_dith;           /* background noise stationarity information*/
      Word16 mem_since_last_sid;    /* number of frames since last SID frame    */

      UWord8 mem_dec_ana_elapsed_count;/* counts elapsed speech frames after DTX*/
      UWord8 mem_dtx_global_state;  /* DTX state flags                         */
      UWord8 mem_data_updated;      /* flags CNI updates                       */
      UWord8 mem_dtx_hangover_count;/* counts down in hangover period          */
      UWord8 mem_sid_frame;         /* flags SID frames                        */
      UWord8 mem_valid_data;        /* flags SID frames containing valid data   */
      UWord8 mem_dtx_hangover_added;/* flags hangover period at end of speech   */

} D_DTX_State;
```

TS26.204-500.doc

Table 6. before change

Table 6: Speech decoder static variables

Struct name	Variable	Type	Length	Description
Decoder_State	mem_gc_thres	Word32	1	Threshold for noise enhancer
	mem_exc	Word16	505	INTERPOL]; /* old excitation vector
	mem_isf_buf	Word16	48	ISF buffer(frequency domain)
	mem_hf	Word16	30	HF band-pass filter memory
	mem_hf2	Word16	30	HF band-pass filter memory
	mem_hf3	Word16	30	HF band-pass filter memory
	mem_oversamp	Word16	24	Synthesis oversampled filter memory
	mem_gain	Word16	23	Gain decoder memory
	mem_syn_hf	Word16	20	HF synthesis memory
	mem_isp	Word16	16	Old ISP (immittance spectral pairs)
	mem_isf	Word16	16	Old ISF (frequency domain)
	mem_isf_q	Word16	16	Past ISF quantizer
	mem_syn_hi	Word16	16	Modified synthesis memory (MSB)
	mem_syn_lo	Word16	16	Modified synthesis memory (LSB)
	mem_ph_disp	Word16	8	Phase dispersion memory
	mem_sig_out	Word16	6	Hp50 filter memory for synthesis
	mem_hp400	Word16	6	Hp400 filter memory for synthesis
	mem_lag	Word16	5	LTP lag history
	mem_subfr_q	Word16	4	Old maximum scaling factor
	mem_tilt_code	Word16	1	Tilt of code
	mem_q	Word16	1	Old scaling factor
	mem_deemph	Word16	1	Speech deemph filter memory
	mem_seed	Word16	1	Random memory for frame erasure
	mem_seed2	Word16	1	Random memory for HF generation
	mem_seed3	Word16	1	Random memory for lag concealment
	mem_T0	Word16	1	Old pitch lag
	mem_T0_frac	Word16	1	Old pitch fraction lag
	mem_vad_hist	UWord16	1	VAD history
	dtx_decSt	D_DTX_State	1	See below in this table
	mem_bfi	UWord8	1	Previous BFI
	mem_state	UWord8	1	BGH state machine memory
	mem_first_frame	UWord8	1	First frame indicator
dtx_decState	mem_isf_buf	Word16	128	ISF vector history (8 frames)
	mem_isf	Word16	16	ISF vector
	mem_isf_prev	Word16	16	Previous ISF vector
	mem_log_en_buf	Word16	8	Logarithmic frame energy history
	mem_true_sid_period_inv	Word16	1	Inverse of true SID update rate
	mem_log_en	Word16	1	Logarithmic frame energy
	mem_log_en_prev	Word16	1	Previous logarithmic frame energy
	mem_cng_seed	Word16	1	Comfort noise excitation seed
	mem_hist_ptr	Word16	1	Index to beginning of LSF history
	mem_dither_seed	Word16	1	Comfort noise dithering seed
	mem_cn_dith	Word16	1	Background noise stationarity information
	mem_dec_ana_elapsed_count	UWord8	1	Counts elapsed speech frames after DTX
	mem_dtx_global_state	UWord8	1	DTX state flags
	mem_since_last_sid	UWord8	1	Number of frames since last SID frame
	mem_data_updated	UWord8	1	Flags CNI updates
	mem_dtx_hangover_count	UWord8	1	Counts down in hangover period
	mem_sid_frame	UWord8	1	Flags SID frames
	mem_valid_data	UWord8	1	Flags SID frames containing valid data
	mem_dtx_hangover_added	UWord8	1	Flags hangover period at end of speech

Table 6. after the change

Table 6: Speech decoder static variables

Struct name	Variable	Type	Length	Description
Decoder_State	mem_gc_thres	Word32	1	Threshold for noise enhancer
	mem_exc	Word16	505	INTERPOL]; /* old excitation vector
	mem_isf_buf	Word16	48	ISF buffer(frequency domain)
	mem_hf	Word16	30	HF band-pass filter memory
	mem_hf2	Word16	30	HF band-pass filter memory
	mem_hf3	Word16	30	HF band-pass filter memory
	mem_oversamp	Word16	24	Synthesis oversampled filter memory
	mem_gain	Word16	23	Gain decoder memory
	mem_syn_hf	Word16	20	HF synthesis memory
	mem_isp	Word16	16	Old ISP (immittance spectral pairs)
	mem_isf	Word16	16	Old ISF (frequency domain)
	mem_isf_q	Word16	16	Past ISF quantizer
	mem_syn_hi	Word16	16	Modified synthesis memory (MSB)
	mem_syn_lo	Word16	16	Modified synthesis memory (LSB)
	mem_ph_disp	Word16	8	Phase dispersion memory
	mem_sig_out	Word16	6	Hp50 filter memory for synthesis
	mem_hp400	Word16	6	Hp400 filter memory for synthesis
	mem_lag	Word16	5	LTP lag history
	mem_subfr_q	Word16	4	Old maximum scaling factor
	mem_tilt_code	Word16	1	Tilt of code
	mem_q	Word16	1	Old scaling factor
	mem_deemph	Word16	1	Speech deemph filter memory
	mem_seed	Word16	1	Random memory for frame erasure
	mem_seed2	Word16	1	Random memory for HF generation
	mem_seed3	Word16	1	Random memory for lag concealment
	mem_T0	Word16	1	Old pitch lag
	mem_T0_frac	Word16	1	Old pitch fraction lag
	mem_vad_hist	UWord16	1	VAD history
	dtx_decSt	D_DTX_State	1	See below in this table
	mem_bfi	UWord8	1	Previous BFI
	mem_state	UWord8	1	BGH state machine memory
	mem_first_frame	UWord8	1	First frame indicator
dtx_decState	mem_isf_buf	Word16	128	ISF vector history (8 frames)
	mem_isf	Word16	16	ISF vector
	mem_isf_prev	Word16	16	Previous ISF vector
	mem_log_en_buf	Word16	8	Logarithmic frame energy history
	mem_true_sid_period_inv	Word16	1	Inverse of true SID update rate
	mem_log_en	Word16	1	Logarithmic frame energy
	mem_log_en_prev	Word16	1	Previous logarithmic frame energy
	mem_cng_seed	Word16	1	Comfort noise excitation seed
	mem_hist_ptr	Word16	1	Index to beginning of LSF history
	mem_dither_seed	Word16	1	Comfort noise dithering seed
	mem_cn_dith	Word16	1	Background noise stationarity information
	mem_since_last_sid	Word16	1	Number of frames since last SID frame
	mem_dec_ana_elapsed_count	UWord8	1	Counts elapsed speech frames after DTX
	mem_dtx_global_state	UWord8	1	DTX state flags
	mem_data_updated	UWord8	1	Flags CNI updates
	mem_dtx_hangover_count	UWord8	1	Counts down in hangover period
	mem_sid_frame	UWord8	1	Flags SID frames
	mem_valid_data	UWord8	1	Flags SID frames containing valid data
	mem_dtx_hangover_added	UWord8	1	Flags hangover period at end of speech

3GPP TSG-SA4 Meeting #25
San Francisco, USA, 20 - 24 January 2003

Tdoc ⌘S4-030087

CR-Form-v7

CHANGE REQUEST

⌘	**26.204**	**CR**	**003**	⌘**rev**	**1**	⌘	Current version:	**5.0.0**	⌘

For ***HELP*** *on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ ☐ ME ☒ Radio Access Network ☐ Core Network ☒

Title:	⌘	Correction to “D_UTIL_pow2” function to be bitexact with TS26.173 counterpart		
Source:	⌘	TSG SA WG4		
Work item code:	⌘	AMRWB-FP	***Date:*** ⌘	18/03/2003
Category:	⌘	**F**	***Release:*** ⌘	Rel-5
		Use one of the following categories: ***F*** *(correction)* ***A*** *(corresponds to a correction in an earlier release)* ***B*** *(addition of feature),* ***C*** *(functional modification of feature)* ***D*** *(editorial modification)* Detailed explanations of the above categories can be found in 3GPP TR 21.900.		*Use one of the following releases:* *2 (GSM Phase 2)* *R96 (Release 1996)* *R97 (Release 1997)* *R98 (Release 1998)* *R99 (Release 1999)* *Rel-4 (Release 4)* *Rel-5 (Release 5)* *Rel-6 (Release 6)*

Reason for change:	⌘	To get TS26.204 decoder bitexact with TS26.173 decoder.
Summary of change:	⌘	D_UTIL_pow2 function is modified so that it is bitexact with the corresponding function in TS26.173
Consequences if not approved:	⌘	TS26.204 decoder is not bitexact with TS26.173 decoder

Clauses affected:	⌘	dec_util.c				
		Y	**N**			
Other specs affected:	⌘		**X**	Other core specifications	⌘	
			X	Test specifications		
			X	O&M Specifications		

1. dec_util.c

Lines 83 to 99 before change:

```
Word32 D_UTIL_pow2(Word16 exponant, Word16 fraction)
{
    Word32 L_x, tmp, i, exp;
    Word16 a;

    L_x = fraction * 32;          /* L_x = fraction<<6             */
    i = L_x >> 15;                /* Extract b10-b16 of fraction   */
    a = (Word16)(L_x);            /* Extract b0-b9   of fraction   */
    a = (Word16)(a & (Word16)0x7fff);
    L_x = D_ROM_pow2[i] << 16;    /* table[i] << 16                */
    tmp = D_ROM_pow2[i] - D_ROM_pow2[i + 1];  /* table[i] - table[i+1] */
    L_x = L_x - ((tmp * a) << 1); /* L_x -= tmp*a*2                */
    exp = 30 - exponant;
    L_x = (L_x + (1 << (exp - 1))) >> exp;

    return(L_x);
}
```

after the change:

```
Word32 D_UTIL_pow2(Word16 exponant, Word16 fraction)
{
    Word32 L_x, tmp, i, exp;
    Word16 a;

    L_x = fraction * 32;          /* L_x = fraction<<6             */
    i = L_x >> 15;                /* Extract b10-b16 of fraction   */
    a = (Word16)(L_x);            /* Extract b0-b9   of fraction   */
    a = (Word16)(a & (Word16)0x7fff);
    L_x = D_ROM_pow2[i] << 16;    /* table[i] << 16                */
    tmp = D_ROM_pow2[i] - D_ROM_pow2[i + 1];  /* table[i] - table[i+1] */
    tmp = L_x - ((tmp * a) << 1); /* L_x -= tmp*a*2                */
    exp = 30 - exponant;
    if (exp <= 31)
    {
    L_x = tmp >> exp;

    if ((1 << (exp - 1)) & tmp)
    {
        L_x++;
    }
    }
    else
    {
    L_x = 0;
    }

    return(L_x);
}
```

3GPP TSG-SA4 Meeting #25
San Francisco, USA, 20-24 January 2003

Tdoc ⌘S4-030088

CR-Form-v7

CHANGE REQUEST

⌘	**26.204**	**CR**	**004**	⌘**rev**	**1**	⌘	Current version:	**5.0.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ □ ME ☒ Radio Access Network □ Core Network ☒

Title:	⌘	MMS compatible i/o format option		
Source:	⌘	TSG SA WG4		
Work item code:	⌘	AMRWB-FP	***Date:*** ⌘	18/03/2003
Category:	⌘	**F**	***Release:*** ⌘	Rel-5

Use one of the following categories:
F *(correction)*
A *(corresponds to a correction in an earlier release)*
B *(addition of feature),*
C *(functional modification of feature)*
D *(editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:

2	*(GSM Phase 2)*
R96	*(Release 1996)*
R97	*(Release 1997)*
R98	*(Release 1998)*
R99	*(Release 1999)*
Rel-4	*(Release 4)*
Rel-5	*(Release 5)*
Rel-6	*(Release 6)*

Reason for change:	⌘	Modifications proposed by this document enable usage of AMR-WB floating-point to encode and decode files according to the AMR-WB MIME file storage format, which is used e.g. by the MMS service.
Summary of change:	⌘	New input/output format option.
Consequences if not approved:	⌘	Codec can not operate with bitstreams in AMR-WB MIME file storage format.

Clauses affected:	⌘	2, 6.3, encoder.c, enc_if.c, if_rom.c, decoder.c, dec_if.c				
		Y	**N**			
Other specs affected:	⌘		**X**	Other core specifications	⌘	
			X	Test specifications		
			X	O&M Specifications		
Other comments:	⌘					

Changes to the specification document:

2. References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TS 26.174: "AMR speech codec, wideband; Test sequences".

[2] 3GPP TS 26.190: "Mandatory Speech Codec speech processing functions AMR Wideband speech codec; Transcoding functions".

[3] 3GPP TS 26.191: "AMR speech codec, wideband; Error concealment of lost frames".

[4] 3GPP TS 26.192: "Mandatory Speech Codec speech processing functions AMR Wideband Speech Codec; Comfort noise aspects".

[5] 3GPP TS 26.193: "AMR speech codec, wideband; Source controlled rate operation".

[6] 3GPP TS 26.194: "Mandatory Speech Codec speech processing functions AMR Wideband speech codec; Voice Activity Detector (VAD)".

[7] RFC 3267 “A Real-Time Transport Protocol (RTP) Payload Format and File Storage Format for Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs, June 2002.

6.3 Parameter bitstream file (encoder output/decoder input)

The files produced by the speech encoder/expected by the speech decoder are described in RFC 3267 [7], sections 5.1 and 5.3

By using a preprocessor definition encoder/decoder can optionally use format described in TS26.201 that defines an octet-aligned frame format (Interface format 2) for the AMR-WB codec.

Changes to the c source-code:

Changes in file *encoder.c*

Lines 12 – 14:

```
#ifndef IF2
#define AMRWB_MAGIC_NUMBER "#!AMR-WB\n"
#endif
```

Lines 62 – 66

```
#ifdef IF2
      fprintf(stderr, "  Described in TS26.201.\n");
#else
      fprintf(stderr, "  Described in RFC 3267 (Sections 5.1 and 5.3).\n");
#endif
```

Lines 127 – 135

```
#ifndef IF2

   /* If MMS output is selected, write the magic number at the beginning of the
    * bitstream file
  */

  fwrite(AMRWB_MAGIC_NUMBER, sizeof(char), strlen(AMRWB_MAGIC_NUMBER), 
f_serial);

#endif
```

Changes in file *enc_if.c*

Lines 130 – 855

```
#ifdef IF2
/*
 * E_IF_if2_conversion
 *
 *
 * Parameters:
 *  mode        I: Mode
 *  param       I: encoder output
 *  stream      O: packed octets (TS26.201)
 *  frame_type  I: TX frame type
 *  dtx         I: speech mode for mode MRDTX
 *
 * Function:
 *  Packing one frame of encoded parameters to AMR-WB IF2
 *
 * Returns:
 *    number of octets
 */
static int E_IF_if2_conversion(Word16 mode, Word16 *param, UWord8 *stream,
                               Word16 frame_type, Word16 speech_mode)
{
   Word32 j = 0;
   Word16 const *mask;

   memset(stream, 0, block_size[mode]);

   switch(mode)
   {
   case MRNO_DATA:
      *stream = 0xF8;
      j = 8;
      break;

   case MODE_7k:
      mask = mode_7k;
      *stream = 0x2;    /* frame_type = 0, fqi = 1  */

      for (j = HEADER_SIZE; j < T_NBBITS_7k; j++)
      {
         if (param[*mask] & *(mask + 1))
         {
            *stream += 0x1;
         }
         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MODE_9k:
      mask = mode_9k;
      *stream = 0x6;    /* frame_type = 1, fqi = 1  */

      for (j = HEADER_SIZE; j < T_NBBITS_9k; j++)
      {
         if (param[*mask] & *(mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MODE_12k:
      mask = mode_12k;
      *stream = 0xA;    /* frame_type = 2, fqi = 1  */

      for (j = HEADER_SIZE; j < T_NBBITS_12k; j++)
      {
         if (param[*mask] & *(mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }
      break;

   case MODE_14k:
      mask = mode_14k;
      *stream = 0xE;    /* frame_type = 3, fqi = 1  */

      for ( j = HEADER_SIZE; j < T_NBBITS_14k; j++ )
      {
         if (param[*mask] & *(mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MODE_16k:
      mask = mode_16k;
      *stream = 0x12;    /* frame_type = 4, fqi = 1  */

      for (j = HEADER_SIZE; j < T_NBBITS_16k; j++)
      {
         if (param[*mask] & *(mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
```

```
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MODE_18k:
      mask = mode_18k;
      *stream = 0x16;    /* frame_type = 5, fqi = 1  */

      for (j = HEADER_SIZE; j < T_NBBITS_18k; j++)
      {
         if (param[*mask] & *(mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MODE_20k:
      mask = mode_20k;
      *stream = 0x1A;    /* frame_type = 6, fqi = 1  */

      for (j = HEADER_SIZE; j < T_NBBITS_20k; j++)
      {
         if (param[*mask] & *( mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MODE_23k:
      mask = mode_23k;
      *stream = 0x1E;    /* frame_type = 7, fqi = 1  */

      for (j = HEADER_SIZE; j < T_NBBITS_23k; j++)
      {
         if (param[*mask] & *( mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MODE_24k:
      mask = mode_24k;
      *stream = 0x22;    /* frame_type = 8, fqi = 1  */

      for (j = HEADER_SIZE; j < T_NBBITS_24k; j++)
      {
         if (param[*mask] & *( mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MRDTX:
      mask = mode_DTX;
      *stream = 0x26;    /* frame_type = 9, fqi = 1  */

      for ( j = HEADER_SIZE; j < T_NBBITS_SID; j++ )
      {
         if (param[*mask] & *(mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }

      }

      /* sid type */
      if (frame_type == TX_SID_UPDATE)
      {
         /* sid update */
         *stream += 0x1;
      }

      /* speech mode indicator */
      *stream <<= 4;
      *stream = (UWord8)(*stream + speech_mode);
      /* bit stuffing */
      *stream <<= 3;
      j = 48;

      break;

   default:
      break;

   }

   return j/8;
}

#else

/*
 * E_IF_mms_conversion
 *
 *
 * Parameters:
 *  mode        I: Mode
 *  param       I: encoder output
 *  stream      O: packed octets (RFC 3267, section 5.3)
 *  frame_type  I: TX frame type
 *  dtx         I: speech mode for mode MRDTX
 *
 * Function:
 *  Packing one frame of encoded parameters to AMR-WB MMS format
 *
 * Returns:
 *    number of octets
 */
static int E_IF_mms_conversion(Word16 mode, Word16 *param, UWord8 *stream,
                               Word16 frame_type, Word16 speech_mode)
{
   Word32 j = 0;
   Word16 const *mask;

   memset(stream, 0, block_size[mode]);

   switch(mode)
   {
   case MRNO_DATA:
      *stream = 0x7C;
      j = 0;
      break;

   case MODE_7k:
      mask = mode_7k;
      *stream = 0x04;    /* frame_type = 0, fqi = 1  */
    stream++;

      for (j = 1; j <= NBBITS_7k; j++)
      {
         if (param[*mask] & *(mask + 1))
         {
            *stream += 0x1;
         }
         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MODE_9k:
      mask = mode_9k;
      *stream = 0x0C;    /* frame_type = 1, fqi = 1  */
    stream++;

      for (j = 1; j <= NBBITS_9k; j++)
      {
         if (param[*mask] & *(mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MODE_12k:
      mask = mode_12k;
      *stream = 0x14;     /* frame_type = 2, fqi = 1  */
    stream++;

      for (j = 1; j <= NBBITS_12k; j++)
      {
         if (param[*mask] & *(mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }
      break;

   case MODE_14k:
      mask = mode_14k;
      *stream = 0x1C;     /* frame_type = 3, fqi = 1  */
    stream++;

      for ( j = 1; j <= NBBITS_14k; j++ )
      {
         if (param[*mask] & *(mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MODE_16k:
      mask = mode_16k;
      *stream = 0x24;     /* frame_type = 4, fqi = 1  */
    stream++;

      for (j = 1; j <= NBBITS_16k; j++)
      {
         if (param[*mask] & *(mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MODE_18k:
      mask = mode_18k;
      *stream = 0x2C;    /* frame_type = 5, fqi = 1  */
    stream++;

      for (j = 1; j <= NBBITS_18k; j++)
      {
         if (param[*mask] & *(mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MODE_20k:
      mask = mode_20k;
      *stream = 0x34;    /* frame_type = 6, fqi = 1  */
    stream++;

      for (j = 1; j <= NBBITS_20k; j++)
      {
         if (param[*mask] & *( mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MODE_23k:
      mask = mode_23k;
      *stream = 0x3C;    /* frame_type = 7, fqi = 1  */
    stream++;

      for (j = 1; j <= NBBITS_23k; j++)
      {
         if (param[*mask] & *( mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MODE_24k:
      mask = mode_24k;
      *stream = 0x44;    /* frame_type = 8, fqi = 1  */
    stream++;

      for (j = 1; j <= NBBITS_24k; j++)
      {
         if (param[*mask] & *( mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }
      while (j % 8)
      {
         *stream <<= 1;
         j++;
      }

      break;

   case MRDTX:
      mask = mode_DTX;
      *stream = 0x4C;     /* frame_type = 9, fqi = 1  */
    stream++;

      for ( j = 1; j <= NBBITS_SID; j++ )
      {
         if (param[*mask] & *(mask + 1))
         {
            *stream += 0x1;
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }

      }

      /* sid type */
      if (frame_type == TX_SID_UPDATE)
      {
         /* sid update */
         *stream += 0x1;
      }

      /* speech mode indicator */
      *stream <<= 4;
      *stream = (UWord8)(*stream + speech_mode);
      j = 40;

      break;

   default:
      break;

   }

   return j/8 + 1;
}

#endif
```

Lines 969 – 973

```
#ifdef IF2
   return E_IF_if2_conversion(mode, prms, serial, frame_type, req_mode);
#else
   return E_IF_mms_conversion(mode, prms, serial, frame_type, req_mode);
#endif
```

Changes in file *if_rom.c*

Lines 75 – 79

```
#ifdef IF2
const UWord8 block_size[16]= {18, 23, 33, 37, 41, 47, 51, 59, 61, 6, 6, 0, 0, 0,
1, 1};
#else
const UWord8 block_size[16]= {18, 24, 33, 37, 41, 47, 51, 59, 61, 6, 6, 0, 0, 0,
1, 1};
#endif
```

Changes in file *decoder.c*

Lines 12 – 15

```
#ifndef IF2
#include <string.h>
#define AMRWB_MAGIC_NUMBER "#!AMR-WB\n"
#endif
```

Lines 44 – 46

```
#ifndef IF2
  char magic[16];
#endif
```

Lines 63 – 67

```
#ifdef IF2
     fprintf(stderr, "  Described in TS26.201.\n");
#else
     fprintf(stderr, "  Described in RFC 3267 (Sections 5.1 and 5.3).\n");
#endif
```

Lines 101 – 113

```
#ifndef IF2
   /* read magic number */
   fread(magic, sizeof(char), strlen(AMRWB_MAGIC_NUMBER), f_serial);

   /* verify magic number */
   if (strncmp(magic, AMRWB_MAGIC_NUMBER, strlen(AMRWB_MAGIC_NUMBER)))
   {
     fprintf(stderr, "%s%s\n", "Invalid magic number: ", magic);
     fclose(f_serial);
     fclose(f_synth);
     exit(0);
   }
#endif
```

Lines 123 – 127

```
#ifdef IF2
       mode = (Word16)(serial[0] >> 4);
#else
     mode = (Word16)((serial[0] >> 3) & 0x0F);
#endif
```

Changes in file *dec_if.c*

Lines 112 – 773

```
#ifdef IF2
/*
 * D_IF_conversion
 *
 *
 * Parameters:
```

```
 *    param             O: AMR parameters
 *    stream            I: input bitstream
 *    frame_type        O: frame type
 *    speech_mode       O: speech mode in DTX
 *    fqi               O: frame quality indicator
 *
 * Function:
 *    Unpacks IF2 octet stream
 *
 * Returns:
 *    mode              used mode
 */
Word16 D_IF_conversion(Word16 *param, UWord8 *stream, UWord8 *frame_type,
                       Word16 *speech_mode, Word16 *fqi)
{
   Word32 mode;
   Word32 j;
   Word16 const *mask;

   memset(param, 0, PRMNO_24k << 1);
   mode = *stream >> 4;
   /* SID indication IF2 corresponds to mode 10 */
   if (mode == 9)
   {
      mode ++;
   }

   *fqi = (Word16)((*stream >> 3) & 0x1);
   *stream <<= (HEADER_SIZE - 1);

   switch (mode)
   {
   case MRDTX:
      mask = mode_DTX;

      for (j = HEADER_SIZE; j < T_NBBITS_SID; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }

         mask += 2;

         if ( j % 8 )
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      /* get SID type bit */

      *frame_type = RX_SID_FIRST;

      if (*stream & 0x80)
      {
         *frame_type = RX_SID_UPDATE;
      }

      *stream <<= 1;

      /* speech mode indicator */
      *speech_mode = (Word16)(*stream >> 4);
      break;

   case MRNO_DATA:
      *frame_type = RX_NO_DATA;
      break;

   case LOST_FRAME:
      *frame_type = RX_SPEECH_LOST;
      break;

   case MODE_7k:
      mask = mode_7k;

      for (j = HEADER_SIZE; j < T_NBBITS_7k; j++)
      {
         if ( *stream & 0x80 )
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }
         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_9k:
      mask = mode_9k;

      for (j = HEADER_SIZE; j < T_NBBITS_9k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }
         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_12k:
      mask = mode_12k;

      for (j = HEADER_SIZE; j < T_NBBITS_12k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }
         mask += 2;

         if ( j % 8 )
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_14k:
      mask = mode_14k;

      for (j = HEADER_SIZE; j < T_NBBITS_14k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }

         mask += 2;

         if ( j % 8 )
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_16k:
      mask = mode_16k;

      for (j = HEADER_SIZE; j < T_NBBITS_16k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_18k:
      mask = mode_18k;

      for (j = HEADER_SIZE; j < T_NBBITS_18k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_20k:
      mask = mode_20k;

      for (j = HEADER_SIZE; j < T_NBBITS_20k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_23k:
      mask = mode_23k;

      for (j = HEADER_SIZE; j < T_NBBITS_23k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }

      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_24k:
      mask = mode_24k;

      for (j = HEADER_SIZE; j < T_NBBITS_24k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }
```

```
         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }

      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   default:
      *frame_type = RX_SPEECH_LOST;
      *fqi = 0;
      break;

   }

   if (*fqi == 0)
   {
      if (*frame_type == RX_SPEECH_GOOD)
      {
         *frame_type = RX_SPEECH_BAD;
      }
      if ((*frame_type == RX_SID_FIRST) | (*frame_type == RX_SID_UPDATE))
      {
         *frame_type = RX_SID_BAD;
      }
   }

   return (Word16)mode;
}
#else

/*
 * D_IF_mms_conversion
 *
 *
 * Parameters:
 *    param                O: AMR parameters
 *    stream               I: input bitstream
 *    frame_type           O: frame type
 *    speech_mode          O: speech mode in DTX
 *    fqi                  O: frame quality indicator
 *
 * Function:
 *    Unpacks MMS formatted octet stream (see RFC 3267, section 5.3)
 *
 * Returns:
 *    mode                 used mode
 */
Word16 D_IF_mms_conversion(Word16 *param, UWord8 *stream, UWord8 *frame_type,
                           Word16 *speech_mode, Word16 *fqi)
{
   Word32 mode;
   Word32 j;
   Word16 const *mask;

   memset(param, 0, PRMNO_24k << 1);

   *fqi = (Word16)((*stream >> 2) & 0x01);
   mode = (Word32)((*stream >> 3) & 0x0F);

   /* SID indication IF2 corresponds to mode 10 */
```

```
    if (mode == 9)
    {
        mode ++;
    }

    stream++;

    switch (mode)
    {
    case MRDTX:
        mask = mode_DTX;

        for (j = 1; j <= NBBITS_SID; j++)
        {
            if (*stream & 0x80)
            {
                param[*mask] = (Word16)(param[*mask] + *(mask + 1));
            }

            mask += 2;

            if ( j % 8 )
            {
                *stream <<= 1;
            }
            else
            {
                stream++;
            }
        }

        /* get SID type bit */

        *frame_type = RX_SID_FIRST;

        if (*stream & 0x80)
        {
            *frame_type = RX_SID_UPDATE;
        }

        *stream <<= 1;

        /* speech mode indicator */
        *speech_mode = (Word16)(*stream >> 4);
        break;

    case MRNO_DATA:
        *frame_type = RX_NO_DATA;
        break;

    case LOST_FRAME:
        *frame_type = RX_SPEECH_LOST;
        break;

    case MODE_7k:
        mask = mode_7k;

        for (j = 1; j <= NBBITS_7k; j++)
        {
            if ( *stream & 0x80 )
            {
                param[*mask] = (Word16)(param[*mask] + *(mask + 1));
            }
            mask += 2;

            if (j % 8)
            {
                *stream <<= 1;
            }
            else
            {
```

```
            stream++;
         }
      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_9k:
      mask = mode_9k;

      for (j = 1; j <= NBBITS_9k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }
         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_12k:
      mask = mode_12k;

      for (j = 1; j <= NBBITS_12k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }
         mask += 2;

         if ( j % 8 )
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_14k:
      mask = mode_14k;

      for (j = 1; j <= NBBITS_14k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }

         mask += 2;

         if ( j % 8 )
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_16k:
      mask = mode_16k;

      for (j = 1; j <= NBBITS_16k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_18k:
      mask = mode_18k;

      for (j = 1; j <= NBBITS_18k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_20k:
      mask = mode_20k;

      for (j = 1; j <= NBBITS_20k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }

         mask += 2;
         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }
      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_23k:
      mask = mode_23k;

      for (j = 1; j <= NBBITS_23k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }

      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   case MODE_24k:
      mask = mode_24k;

      for (j = 1; j <= NBBITS_24k; j++)
      {
         if (*stream & 0x80)
         {
            param[*mask] = (Word16)(param[*mask] + *(mask + 1));
         }

         mask += 2;

         if (j % 8)
         {
            *stream <<= 1;
         }
         else
         {
            stream++;
         }

      }

      *frame_type = RX_SPEECH_GOOD;
      break;

   default:
      *frame_type = RX_SPEECH_LOST;
      *fqi = 0;
      break;

   }
   if (*fqi == 0)
   {
      if (*frame_type == RX_SPEECH_GOOD)
      {
         *frame_type = RX_SPEECH_BAD;
      }
      if ((*frame_type == RX_SID_FIRST) | (*frame_type == RX_SID_UPDATE))
      {
         *frame_type = RX_SID_BAD;
      }
   }

   return (Word16)mode;
}

#endif
```

Lines 815 – 828

```
#ifdef IF2
      *bits = (UWord8)((Word32)*bits & ~(lfi << 3));
#else
      *bits = (UWord8)((Word32)*bits & ~(lfi << 2));
#endif
      /*
       * extract mode information and frame_type,
       * octets to parameters
       */
#ifdef IF2
      mode = D_IF_conversion( prm, bits, &frame_type, &speech_mode, &fqi);
#else
      mode = D_IF_mms_conversion( prm, bits, &frame_type, &speech_mode, &fqi);
#endif
```

3GPP TSG-SA4 Meeting #25bis
Berlin, Germany, 24-28 February 2003

Tdoc ⌘S4-030144

CR-Form-v7

CHANGE REQUEST

⌘	**26.204**	**CR**	**005**	⌘**rev**	**-**	⌘	Current version:	**5.0.0**	⌘

For ***HELP*** *on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ □ ME ☒ Radio Access Network □ Core Network ☒

Title:	⌘	Correction for handling of RX_NO_DATA frames		
Source:	⌘	TSG SA WG4		
Work item code:	⌘	AMRWB-FP	***Date:*** ⌘	18/03/2003
Category:	⌘	**F** *Use one of the following categories:* ***F*** *(correction)* ***A*** *(corresponds to a correction in an earlier release)* ***B*** *(addition of feature),* ***C*** *(functional modification of feature)* ***D*** *(editorial modification)* Detailed explanations of the above categories can be found in 3GPP TR 21.900.	***Release:*** ⌘	Rel-5 *Use one of the following releases:* *2 (GSM Phase 2)* *R96 (Release 1996)* *R97 (Release 1997)* *R98 (Release 1998)* *R99 (Release 1999)* *Rel-4 (Release 4)* *Rel-5 (Release 5)* *Rel-6 (Release 6)*

Reason for change:	⌘	Frames of type RX_NO_DATA are not handled appropriately in the AMR-WB speech decoder. The flag “unusable_frame” is set to false for this frametype although RX_NO_DATA frames do not contain any useful information.
Summary of change:	⌘	For frames of type RX_NO_DATA, the flag “unusable_frame” is set to true now.
Consequences if not approved:	⌘	Possible speech degradations if NO_DATA frames are received without preceding SID_FIRST frame.

Clauses affected:	⌘	Source file: dec_main.c				
		Y	**N**			
Other specs affected:	⌘		**X**	Other core specifications	⌘	
			X	Test specifications		
			X	O&M Specifications		
Other comments:	⌘					

1. How the code is changed in the file *dec_main.c*

Lines 245-263:

```
/* SPEECH action state machine  */
if((frame_type == RX_SPEECH_BAD) | ~~(frame_type == RX_NO_DATA) |~~
   (frame_type == RX_SPEECH_PROBABLY_DEGRADED))
{
   /* bfi for all index, bits are not usable */
   bfi = 1;
   unusable_frame = 0;
}
else if((frame_type == RX_NO_DATA) |(frame_type == RX_SPEECH_LOST))
{
   /* bfi only for lsf, gains and pitch period */
   bfi = 1;
   unusable_frame = 1;
}
else
{
   bfi = 0;
   unusable_frame = 0;
}
```

3GPP TSG-SA4 Meeting #25bis
Berlin, Germany, 24-28 February 2003

Tdoc ⌘S4-030174

CR-Form-v7

CHANGE REQUEST

⌘	**26.204**	**CR**	**006**	⌘ **rev**	**1**	⌘	Current version:	**5.0.0**	⌘	

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ □ ME ☒ Radio Access Network □ Core Network ☒

Title:	⌘	Ambiguous expressions in the AMR-WB Floating-point C-Code		
Source:	⌘	TSG SA WG4		
Work item code:	⌘	AMRWB-FP	***Date:*** ⌘	18/3/2003
Category:	⌘	**F**	***Release:*** ⌘	Rel-5

Use one of the following categories:
F *(correction)*
A *(corresponds to a correction in an earlier release)*
B *(addition of feature),*
C *(functional modification of feature)*
D *(editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:

2	*(GSM Phase 2)*
R96	*(Release 1996)*
R97	*(Release 1997)*
R98	*(Release 1998)*
R99	*(Release 1999)*
Rel-4	*(Release 4)*
Rel-5	*(Release 5)*
Rel-6	*(Release 6)*

Reason for change:	⌘	The C-code is ambiguous. ANSI C does not define how to evaluate the following kind of expression: *p0++ = *p0 * psign[j]. The subexpression p0++ causes a side effect which leads to undefined behavior since p0 is also referenced elsewhere in the same expression.
Summary of change:	⌘	Ambiguous expressions are replaced by non-ambiguous expressions reflecting the intention of the author.
Consequences if not approved:	⌘	The reference C-code is ambiguous. Some compilers may produce incorrect binary code and test vectors may fail.

Clauses affected:	⌘	Source file: enc_acelp.c				
		Y	**N**			
Other specs affected:	⌘		X	Other core specifications	⌘	
			X	Test specifications		
			X	O&M Specifications		
Other comments:	⌘					

1. How the code is changed in the file *enc_acelp.c* function *E_ACELP_2t*

1.1 Before the change

Lines 1003-1006 read:

```
for(j = 1; j < L_SUBFR; j += 2)
      {
         *p0++ = *p0 * psign[j];
      }
```

1.2 After the change

Lines read now:

```
for(j = 1; j < L_SUBFR; j += 2)
      {
         *p0 = *p0 * psign[j];
          p0++;
      }
```

2. How the code is changed in the file *enc_acelp.c* function *E_ACELP_4t*

2.1 Before the change

Lines 1496-1511 read:

```
*p0++ = *p0 * psign[j];
*p0++ = *p0 * psign[j + 4];
*p0++ = *p0 * psign[j + 8];
*p0++ = *p0 * psign[j + 12];
*p0++ = *p0 * psign[j + 16];
*p0++ = *p0 * psign[j + 20];
*p0++ = *p0 * psign[j + 24];
*p0++ = *p0 * psign[j + 28];
*p0++ = *p0 * psign[j + 32];
*p0++ = *p0 * psign[j + 36];
*p0++ = *p0 * psign[j + 40];
*p0++ = *p0 * psign[j + 44];
*p0++ = *p0 * psign[j + 48];
*p0++ = *p0 * psign[j + 52];
*p0++ = *p0 * psign[j + 56];
*p0++ = *p0 * psign[j + 60];
```

2.2 After the change

Lines now read:

```
p0[0] = p0[0] * psign[j];
p0[1] = p0[1] * psign[j + 4];
p0[2] = p0[2] * psign[j + 8];
p0[3] = p0[3] * psign[j + 12];
p0[4] = p0[4] * psign[j + 16];
p0[5] = p0[5] * psign[j + 20];
p0[6] = p0[6] * psign[j + 24];
p0[7] = p0[7] * psign[j + 28];
p0[8] = p0[8] * psign[j + 32];
p0[9] = p0[9] * psign[j + 36];
p0[10] = p0[10] * psign[j + 40];
p0[11] = p0[11] * psign[j + 44];
p0[12] = p0[12] * psign[j + 48];
p0[13] = p0[13] * psign[j + 52];
p0[14] = p0[14] * psign[j + 56];
p0[15] = p0[15] * psign[j + 60];
p0 += 16;
```