Technical Specification Group Services and System Aspects **TSGS#21(03)0446**
Meeting #21, Frankfurt, Germany
22-25 September 2003

Source: **TSG-SA WG4**

Title: CRs to TS 26.173 - Possible decoder LPC coefficients overflow (Release 5)

Document for: **Approval**

Agenda Item: **7.4.3**

The following CR, agreed at the TSG-SA WG4 meeting #28, is presented to TSG SA #21 for approval.

Spec	CR	Rev	Phase	Subject	Cat	Vers	WG	Meeting	S4 doc
26.173	019		Rel-5	Possible decoder LPC coefficients overflow	F	5.7.1	S4	TSG-SA WG4#28	S4-030634

3GPP TSG-SA4 Meeting #28
Erlangen, Germany, 1-5 Sep 2003

Tdoc ⌘S4-030634

CR-Form-v7

CHANGE REQUEST

⌘ **26.173**	**CR 019**	⌘ **rev**	**-**	⌘ Current version:	**5.7.1** ⌘

*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 [X] Radio Access Network [ ] Core Network [X]

Title: ⌘	Possible decoder LPC coefficients overflow		
Source: ⌘	TSG SA WG4		
Work item code: ⌘	AMRWB	***Date:*** ⌘	22/9/2003
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: ⌘	AMR-WB decoder can produce unstable output during DTX-operation.
Summary of change: ⌘	Conversion from ISP to LPC coefficients is changed, so that LPC coefficients cannot overflow in decoder comfort noise generation. Synthesis is changed to support scaled LPC coefficients.
Consequences if not approved: ⌘	Decoder synthesis filter may be unstable causing uncontrolled ouput when DTX is used.

Clauses affected: ⌘	acelp.h, cod_main.c, dec_main.c, int_lpc.c, lsp_az.c and syn_filt.c			
	Y	**N**		
Other specs affected: ⌘		**X**	Other core specifications	⌘
		X	Test specifications	
		X	O&M Specifications	
Other comments: ⌘	Example of LPC coefficients saturation. Input signal is processed with AMR-WB encoder using mode 7 and DTX on. Following figures shows decoder output using v5.7.0 decoder and fixed decoder with adaptive scaling of LPC coefficients.			

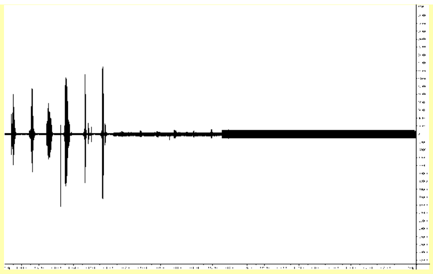

Figure 1 Input signal

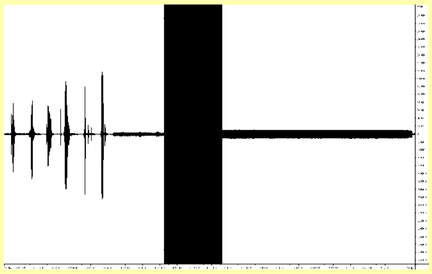

Figure 2 Decoder output v5.7.0

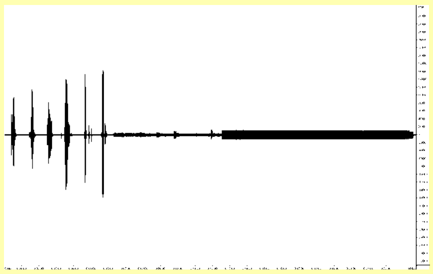

Figure 3 Decoder with adaptive LPC scaling

Overhead from additional operations is listed below.

encoder/decoder	VAD	mode(s)	wOPS
Decoder	on	0	71200
Decoder	on	1-8	51400
Decoder	off	0-8	74600
Encoder	on	0-8	73800
Encoder	off	0-8	55350

According to the design constraints for the AMR-WB speech codec up to 41.6 wMOPS were allowed including the VAD/DTX system (see Tdoc S4-000340: "Permanent Project Document: Design Constraints (WB-4, version 1.3)", 3GPP TSG-S4). The measured AMR-WB TWC figure is 38.97 wMOPS (Tdoc S4-010008). Maximum TWC is increased if adaptive LPC scaling is used by 0.0738 wMOPS (Encoder mode 7 vad on) + 0.0712 wMOPS (Decoder mode 0 vad on). This is clearly below design constraints limit.

Changes to the C-code:

1. How the code is changed in the file *acelp.h*

Lines 66 - 70:

```
void Isp_Az(
     Word16 isp[],                          /* (i) Q15 : Immittance spectral pairs            */
     Word16 a[],                            /* (o) Q12 : predictor coefficients (order = M)   */
     Word16 m
m,
     Word16 adaptive_scaling                /* (i) 0    : adaptive scaling disabled */
                                            /*     1    : adaptive scaling enabled  */
);
```

2. How the code is changed in the file *cod_main.c*

Lines 636 – 638:

```
/* Convert ISFs to the cosine domain */
        Isf_isp(isf, ispnew_q, M);
        Isp_Az(ispnew_q, Aq, M, 0);
```

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

Lines 319 – 324:

```
/* Convert ISFs to the cosine domain */
        Isf_isp(isf, ispnew, M);

        Isp_Az(ispnew, Aq, M, 1);

        Copy(st->isfold, isf_tmp, M);
```

Lines 1114 – 1122:

```
test();test();
    if ((sub(nb_bits, NBBITS_7k) <= 0) && (sub(newDTXState, SPEECH) == 0))
    {
        Isf_Extrapolation(HfIsf);
        Isp_Az(HfIsf, HfA, M16k, 0);

        Weight_a(HfA, Ap, 29491, M16k);     /* fac=0.9 */
        Syn_filt(Ap, M16k, HF, HF, L_SUBFR16k, st->mem_syn_hf, 1);
    } else
```

4. How the code is changed in the file *int_lpc.c*

Lines 16 – 47:

```
void Int_isp(
     Word16 isp_old[],                      /* input : isps from past frame              */
     Word16 isp_new[],                      /* input : isps from present frame           */
     Word16 frac[],                         /* input : fraction for 3 first subfr (Q15)  */
     Word16 Az[]                            /* output: LP coefficients in 4 subframes    */
)
{
    Word16 i, k, fac_old, fac_new;
    Word16 isp[M];
    Word32 L_tmp;
```

```
    for (k = 0; k < 3; k++)
    {
        fac_new = frac[k];                move16();
        fac_old = add(sub(32767, fac_new), 1);  /* 1.0 - fac_new */

        for (i = 0; i < M; i++)
        {
            L_tmp = L_mult(isp_old[i], fac_old);
            L_tmp = L_mac(L_tmp, isp_new[i], fac_new);
            isp[i] = round(L_tmp);        move16();
        }
        Isp_Az(isp, Az, M, 0);
        Az += MP1;
    }

    /* 4th subframe: isp_new (frac=1.0) */

    Isp_Az(isp_new, Az, M, 0);

    return;
}
```

5. How the code is changed in the file *isp_az.c*

Lines 21-119:

```
void Isp_Az(
     Word16 isp[],                        /* (i) Q15 : Immittance spectral pairs          */
     Word16 a[],                          /* (o) Q12 : predictor coefficients (order = M)   */
     Word16 m
m,
     Word16 adaptive_scaling              /* (i) 0    : adaptive scaling disabled */
                                          /*     1    : adaptive scaling enabled  */

)
{
    Word16 i, j, hi, lo;
    Word32 f1[NC16k + 1], f2[NC16k];
    Word16 nc;
    Word32 t0;
    Word16 q, q_sug;
    Word32 tmax;


    nc = shr(m, 1);
    test();
    if (sub(nc, 8) > 0)
    {
        Get_isp_pol_16kHz(&isp[0], f1, nc);
        for (i = 0; i <= nc; i++)
        {
            f1[i] = L_shl(f1[i], 2);      move32();
        }
    } else
        Get_isp_pol(&isp[0], f1, nc);

    test();
    if (sub(nc, 8) > 0)
    {
        Get_isp_pol_16kHz(&isp[1], f2, sub(nc, 1));
        for (i = 0; i <= nc - 1; i++)
        {
            f2[i] = L_shl(f2[i], 2);      move32();
        }
    } else
        Get_isp_pol(&isp[1], f2, sub(nc, 1));

    /*-----------------------------------------------------*
     *  Multiply F2(z) by (1 - z^-2)                       *
     *-----------------------------------------------------*/

    for (i = sub(nc, 1); i > 1; i--)
    {
        f2[i] = L_sub(f2[i], f2[i - 2]);   move32();  /* f2[i] -= f2[i-2]; */
    }
```

```
    /*---------------------------------------------------------*
     *  Scale F1(z) by (1+isp[m-1])  and  F2(z) by (1-isp[m-1]) *
     *---------------------------------------------------------*/

    for (i = 0; i < nc; i++)
    {
        /* f1[i] *= (1.0 + isp[M-1]); */

        L_Extract(f1[i], &hi, &lo);
        t0 = Mpy_32_16(hi, lo, isp[m - 1]);
        f1[i] = L_add(f1[i], t0);          move32();

        /* f2[i] *= (1.0 - isp[M-1]); */

        L_Extract(f2[i], &hi, &lo);
        t0 = Mpy_32_16(hi, lo, isp[m - 1]);
        f2[i] = L_sub(f2[i], t0);          move32();
    }

    /*---------------------------------------------------*
     *  A(z) = (F1(z)+F2(z))/2                           *
     *  F1(z) is symmetric and F2(z) is antisymmetric    *
     *---------------------------------------------------*/

    /* a[0] = 1.0; */
    a[0] = 4096;                           move16();
    tmax = 1;                              move32();


    for (i = 1, j = sub(m, 1); i < nc; i++, j--)
    {
        /* a[i] = 0.5*(f1[i] + f2[i]); */

        t0 = L_add(f1[i], f2[i]);          /* f1[i] + f2[i]             */
        tmax |= L_abs(t0);                 logic32();

        a[i] = extract_l(L_shr_r(t0, 12)); /* from Q23 to Q12 and * 0.5 */
        move16();

        /* a[j] = 0.5*(f1[i] - f2[i]); */

        t0 = L_sub(f1[i], f2[i]);          /* f1[i] - f2[i]             */
        tmax |= L_abs(t0);                 logic32();

        a[j] = extract_l(L_shr_r(t0, 12)); /* from Q23 to Q12 and * 0.5 */
        move16();
    }

    /* rescale data if overflow has occured and reprocess the loop */

    test();
    if ( sub(adaptive_scaling, 1) == 0 )
       q = sub(4, norm_l(tmax));        /* adaptive scaling enabled */
    else
       q = 0;           move16();       /* adaptive scaling disabled */

    test();
    if (q > 0)
    {
      q_sug = add(12, q);
      for (i = 1, j = sub(m, 1); i < nc; i++, j--)
        {
          /* a[i] = 0.5*(f1[i] + f2[i]); */

          t0 = L_add(f1[i], f2[i]);          /* f1[i] + f2[i]             */
          a[i] = extract_l(L_shr_r(t0, q_sug)); /* from Q23 to Q12 and * 0.5 */
          move16();

          /* a[j] = 0.5*(f1[i] - f2[i]); */

          t0 = L_sub(f1[i], f2[i]);          /* f1[i] - f2[i]             */
          a[j] = extract_l(L_shr_r(t0, q_sug)); /* from Q23 to Q12 and * 0.5 */
          move16();
        }
      a[0] = shr(a[0], q);                 move16();
    }
    else
    {
      q_sug = 12;                          move16();
      q     = 0;                           move16();
```

```
    }


    /* a[NC] = 0.5*f1[NC]*(1.0 + isp[M-1]); */

    L_Extract(f1[nc], &hi, &lo);
    t0 = Mpy_32_16(hi, lo, isp[m - 1]);
    t0 = L_add(f1[nc], t0);
    a[nc] = extract_l(L_shr_r(t0, ~~12));~~q_sug));    /* from Q23 to Q12 and * 0.5 */
    move16();
    /* a[m] = isp[m-1]; */

    a[m] = shr_r(isp[m - 1], ~~3);~~add(3,q));           /* from Q15 to Q12          */
    move16();

    return;
}
```

6. How the code is changed in the file *syn_filt.c*

Lines 14-107:

```
void Syn_filt(
     Word16 a[],                           /* (i) Q12 : a[m+1] prediction coefficients           */
     Word16 m,                             /* (i)     : order of LP filter                       */
     Word16 x[],                           /* (i)     : input signal                             */
     Word16 y[],                           /* (o)     : output signal                            */
     Word16 lg,                            /* (i)     : size of filtering                        */
     Word16 mem[],                         /* (i/o)   : memory associated with this filtering.   */
     Word16 update                         /* (i)     : 0=no update, 1=update of memory.         */
)
{
    Word16 i, j, y_buf[L_SUBFR16k + M16k], a0, s;
    Word32 L_tmp;
    Word16 *yy;

    yy = &y_buf[0];                       move16();

    /* copy initial filter states into synthesis buffer */
    for (i = 0; i < m; i++)
    {
        *yy++ = mem[i];                   move16();
    }

    s = sub(norm_s(a[0]), 2);

    a0 = shr(a[0], 1);                    /* input / 2 */

    /* Do the filtering. */

    for (i = 0; i < lg; i++)
    {
        L_tmp = L_mult(x[i], a0);

        for (j = 1; j <= m; j++)
            L_tmp = L_msu(L_tmp, a[j], yy[i - j]);

        L_tmp = L_shl(L_tmp, ~~3);~~add(3, s));

        y[i] = yy[i] = round(L_tmp);      move16();move16();
    }

    /* Update memory if required */
    test();
    if (update)
        for (i = 0; i < m; i++)
        {
            mem[i] = yy[lg - m + i];      move16();
        }

    return;
}


void Syn_filt_32(
     Word16 a[],                           /* (i) Q12 : a[m+1] prediction coefficients */
```

```
     Word16 m,                        /* (i)      : order of LP filter             */
     Word16 exc[],                    /* (i) Qnew: excitation (exc[i] >> Qnew)     */
     Word16 Qnew,                     /* (i)      : exc scaling = 0(min) to 8(max) */
     Word16 sig_hi[],                 /* (o) /16 : synthesis high                  */
     Word16 sig_lo[],                 /* (o) /16 : synthesis low                   */
     Word16 lg                        /* (i)      : size of filtering              */
)
{
    Word16 i, j, a0, s;
    Word32 L_tmp;

    s = sub(norm_s(a[0]), 2);


    a0 = shr(a[0], add(4, Qnew));          /* input / 16 and >>Qnew */

    /* Do the filtering. */

    for (i = 0; i < lg; i++)
    {
        L_tmp = 0;                         move32();
        for (j = 1; j <= m; j++)
            L_tmp = L_msu(L_tmp, sig_lo[i - j], a[j]);

        L_tmp = L_shr(L_tmp, 16 - 4);      /* -4 : sig_lo[i] << 4 */

        L_tmp = L_mac(L_tmp, exc[i], a0);

        for (j = 1; j <= m; j++)
            L_tmp = L_msu(L_tmp, sig_hi[i - j], a[j]);

        /* sig_hi = bit16 to bit31 of synthesis */
        L_tmp = L_shl(L_tmp, 3);add(3, s));           /* ai in Q12 */
        sig_hi[i] = extract_h(L_tmp);      move16();

        /* sig_lo = bit4 to bit15 of synthesis */
        L_tmp = L_shr(L_tmp, 4);           /* 4 : sig_lo[i] >> 4 */
        sig_lo[i] = extract_l(L_msu(L_tmp, sig_hi[i], 2048));   move16();
    }

    return;
}
```