รายการอ้างอิง

ADSP-2100 family assembler tools & simulator manual. 2nd ed. Norwood, MA : Analog Devices, 1994.

ADSP-2100 family user's manual. Eaglewood Cliffs, NJ : Prentice-Hall, 1994.

Degener, J. Putting the GSM 06.10 RPE-LTP algorithm to work [machine readable data file]. Martinez, CA : Dr. Dobb's Journal, 1995.

Degener, J., and Bormann, C. GSM 06.10 13 kbit/s RPE/LTP speech compression [computer program]. Berlin : Technische Universitaet Berlin, 1992.

Deller, J., R., Jr., Proakis, J., G., and Hansen, J., H.L. Discrete-time processing of speech signals. New York : Macmillan Publishing, 1993.

Finke, M., Gaspard, M., Ringstrom, T., Savioli, S., and Weistrom, M. http://129.16.30.11/~d2rinto/Reports/SpeechCompression/SpeechCompression.html. 1996.

Frerking, M., E. Digital signal processing in communication systems. New York : Van Nostrand Reinhold, 1994.

Kondos, A., M. Digital speech coding for low bit rate communication systems. Chichester, UK : John Wiley & Sons, 1995.

Mar, A., ed. Digital signal processing applications using the ADSP-2100 family. Vol. 1. New Jersey : Prentice-Hall, 1992.

Nath, N.S., M. Leading Edge. EDN Asia 10 (October 1996) : 17.

Pohlmann, K., C. Advanced digital audio. Indiana : Sams, 1991.

U.S. Department of Defense. CELP voice coder version 3.2c [computer program]. n.p. : n.d.

U.S. Department of Defense. LPC-10 2400 bps voice coder [computer program]. n.p. : 1993.

ภาคผนวก

บางส่วนของโปรแกรมภาษาแอสเซมบลีเอดีเอสพี2101 สำหรับวิเคราะห์สัญญาณเสียงในวิธีอาร์พีอี-แอลทีพี

```
.MODULE Correlation;
.VAR/DM maxcc_scale;
.INCLUDE <lpcconst.h>;
.GLOBAL maxcc_scale;
{
        Correlate Routine
        Desc:       Calculate correlation of input signal

        Calling Parameters
                I1 --> Data Memory Buffer                  L1 = 0
                I2 --> Length of Data Buffer               L2 = 0
                I5 --> Program Memory Buffer               L5 = 0
                I6 --> Program Memory Result Buffer     L6 = 0
                M1,M5 = 1
                M2 = -1
                SE = scale value
                CNTR = output buffer length

        Return Values
                Result Buffer Filled

}

.ENTRY          correlate;
{----------------------------------}
correlate:
{----------------------------------}

                AY0=1;
                DO corr_loop UNTIL CE;
                   I0=I1;                          { restore I0 from I1 }
                   I4=I5;                          { restore I4 from I5 }
                   CNTR=AR;                        { restore loop counter }
                   MR=0, MY0=PM(I4,M5), MX0=DM(I0,M1);
                   DO data_loop UNTIL CE;
data_loop:            MR=MR+MX0*MY0(SS),MY0=PM(I4,M5),MX0=DM(I0,M1);
                   MY0=PM(I5,M5),AR=AR-AY0;        {I5=I5+1, loop counter -= 1 }
                   DM(I2,M1)=MR0;
corr_loop:         PM(I6,M5)=MR1;                  { save result }
                RTS;


{
    schur.dsp
    Desc:       Calculate reflection coefficients from input autocorrelation
    Author:     Meelarp R. C618273
```

```
    Date:          Mar 30, 96
    Update:     Revise from 4.12 format to 1.15 format May 4,96
}

.MODULE  Schur;
{
    This routine computes the LPC reflection coefficients for the input data
     using Schur algorithm.

    Calling Parameters
        I4 --> Autocorrelation Buffer
        I0 --> Reflection Coefficients

    Return Values
        Output Buffer filled
}

.INCLUDE    <lpcconst.h>;
.INCLUDE    <divide.mac>;
.VAR/DM     K[P+1],PP[P+1];
.VAR/DM     error, n;

.ENTRY      schur;
schur:
            I5=I4;
            AR=PM(I5,M5);
            AR=PASS AR;
            IF NE JUMP acf0_is_not_zero;
            AR=PM(I5,M5);
            AR=PASS AR;
            IF NE JUMP acf0_is_not_zero;
acf0_is_zero:
            CNTR=P;
            AR=0;
            DO clear_ref_coeff UNTIL CE;
clear_ref_coeff:
            DM(I0,M1)=AR;
            RTS;

acf0_is_not_zero:

            /* l_acf must have been normalized already */

            /* initialize K[] for the recursion */
            CNTR=P-1;
            I1=^K+1;
            I5=I4;
            MODIFY(I5,M5);
            DO init_k_array UNTIL CE;
                AR=PM(I5,M5);
init_k_array:
              DM(I1,M1)=AR;
            CNTR=P+1;
            I1=^PP;
            I5=I4;
            DO init_p_array UNTIL CE;
```

```
               AR=PM(I5,M5);
init_p_array:
               DM(I1,M1)=AR;

            /* Compute reflection coefficients */
            AR=1;
            DM(n)=AR;                  { n=1 }
            CNTR=P;
            DO compute_ref_coeff UNTIL CE;
                I1=^PP+1;
                AR=DM(I1,M2);
                AR=ABS AR;
                AY1=AR;
                AX1=DM(I1,M0);
                AR=AX1-AY1;
                IF GE JUMP p0_ge_p1;
                    AX0=P+1;
                    AY0=DM(n);
                    AR=AX0-AY0;
                    CNTR=AR;                   { for(i=n; i<=P; i++  ... }
                    AR=0;
                    DO clear_ref_coeff_and_rts UNTIL CE;
clear_ref_coeff_and_rts:
                        DM(I0,M1)=AR;
                    POP CNTR, POP LOOP, POP PC;
                    RTS;
p0_ge_p1:
                AY0=0;                         { AY1:AY0 = ABS(PP[1]) }

                AX0=AX1;                       { AX0 = PP[0] }
                divide(AX0,AY1);               { AY1:AY0 / AX0 }
                I1=^PP+1;
                AR=DM(I1,M0);
                AR=PASS AR;
                IF LE JUMP p1_le_zero;         { if(PP[1] > 0 ) *r = -*r }
                    AR=-AY0;
                    AY0=AR;
p1_le_zero:
                DM(I0,M0)=AY0;                 { *r = gsm_div() ... }

                AX0=DM(n);                     { if( n== 8 ) return }
                AY0=8;
                AR=AX0-AY0;
                IF EQ JUMP return_from_loop;

                /* schur recursion */
                I1=^PP+1;
                MX0=DM(I1,M2);                { MX0 = PP[1] }
                MY0=DM(I0,M0);                { MY0 = *r }
                AX0=DM(I1,M0);                { AX0 = PP[0] }
                MR=MX0*MY0(SS);
                MR=MR(RND);
                AY0=MR1;
                AR=AX0+AY0;
                DM(I1,M0)=AR;
```

```
                  AX0=P;
                  AY0=DM(n);
                  AR=AX0-AY0;
                  CNTR=AR;
                  I1=^K+1;
                  I2=^PP+2;
                  DO schur_recursion UNTIL CE;
                      MX0=DM(I1,M0);
                      MY0=DM(I0,M0);
                      AX0=DM(I2,M2);
                      MR=MX0*MY0(SS);
                      MR=MR(RND);
                      AY0=MR1;
                      AR=AX0+AY0;
                      DM(I2,M1)=AR;           { PP[m] = ... }

                      AX0=DM(I1,M0);          { AX0=K[m] }
                      MX0=DM(I2,M1);
                      MY0=DM(I0,M0);          { MX0=PP[m+1], MY0 = *r }
                      MR=MX0*MY0(SS);
                      MR=MR(RND);
                      AY0=MR1;
                      AR=AX0+AY0;
                      DM(I1,M1)=AR;
schur_recursion:      NOP;


                  AY0=DM(n);                  { perform n++ }
                  AR=AY0+1;
                  DM(n)=AR;
                  MODIFY(I0,M1);              { perform r++ }
compute_ref_coeff:
                  NOP;
                  RTS;

return_from_loop:
                  POP CNTR, POP LOOP, POP PC;
                  RTS;
.ENDMOD;


.MODULE ShortTermAnalysis;
{
   Short Term Analysis Routine
   Calling Parameters
      I0 --> Point to Input Samples                 double const *in
      I1 --> Point to LPC coefficients              double const *ref
      I2 --> Point to Output Result in Data Meory     double *out
         I4 --> LENGTH;
   Return Values
      Result Buffer Filled
   Author
      Meelarp R.
}
```

```
.INCLUDE <lpcconst.h>;

.VAR/PM/RAM    u[P];
.VAR/DM/RAM    my_cntr;
.GLOBAL           my_cntr;
.INIT                u:<u.dat>;
.ENTRY            short_term_anal;

short_term_anal:
              M0=0;M1=1;M4=0;M5=1;
              SI=I1;                  { save I1 }
                      DM(my_cntr)=I4;
st_anal_big_loop:
                 AY0=DM(I0,M1);       { sav = s = *in++ }
                 MX1=AY0;             { AY0 = sav, MX1 = s }
                 I4=^u;
                 I1=SI;               { restore I1 }
                 CNTR=P;
                 DO st_anal_repeat UNTIL CE;
                              AR=PM(I4,M4);     { AR = ui = u[i] }
                    PM(I4,M5)=AY0; { u[i] = sav }
                              MR1=AR;
                              MR0=0; MX0=DM(I1,M1); MY0=MX1;
                    { MR = ui, MX0 = ref[i], MY0 = s }
                    MR=MR+MX0*MY0(SS);
                    { MR = ui + ref[i] * s }
                              MR=MR(RND);
                              AY0=MR1;
                    { sav = ui + ref[i] * s }
                              MR1=MX1;
                              MR0=0; MY0=AR;
                    { MR = s, MX0 = ref[i], MY0 = ui }
                    MR=MR+MX0*MY0(SS);
                    { MR = s + ref[i] * ui }
                              MR=MR(RND);
                              MX1=MR1;
                               { s = s + ref[i] * ui }
st_anal_repeat:
                      NOP;
                         NOP;
                         DM(I2,M1)=MX1;

                       AR=DM(my_cntr);          /* read counter into AR */
                       AY0=1;
                       AR=AR-AY0;               /* decrement counter */
                       IF EQ JUMP st_anal_end;  /* counter expires? */
                       DM(my_cntr)=AR;
                         JUMP st_anal_big_loop; /* no, repeat the loop */
st_anal_loop:
st_anal_end:      NOP;
               RTS;
.ENDMOD;


{
```

```
    crosscor.dsp
    Desc:       Calculate crosscorrelation of input signal
    Author:     Meelarp R. C618273
    Date:          Mar 30,96
}

.MODULE CrossCorrelation;

{
        Cross Correlate Routine

        Parameter Setup
                I1 --> Data Memory Buffer                      --> double const *
x
                I5 --> Program Memory Buffer                   --> double const *
y
                I6 --> Program Memory Result Buffer(msw)       --> double * c
                I0 --> Data Memory Result Buffer (lsw)         --> double * c
                M1,M5 = 1
                M2,M6 = -1
                AY0 = output buffer length                     --> int lag = 2*
SUBWIN

        Return Values
                Result Buffer Filled

        Altered Registers

        Computation Time
}

.INCLUDE      <lpcconst.h>;
.EXTERNAL  prev, ccl, cc;
.ENTRY        crosscorrelate;

crosscorrelate:
                AY0=2*SUBWIN;
                I6=^cc+2*SUBWIN-1;
                I0=^ccl+2*SUBWIN-1;
                I5=^prev+1;
                CNTR=2*SUBWIN;                   { CNTR = lag }
                DO crosscorr_loop UNTIL CE;
                    I2=I1;                       { restore I2 from I1 }
                    I4=I5;                       { restore I4 from I5 }
                    CNTR=SUBWIN;                 { restore loop counter }
                    MR=0, MY0=PM(I4,M5), MX0=DM(I2,M1);
                    DO data_loop UNTIL CE;
data_loop:              MR=MR+MX0*MY0(SS),MY0=PM(I4,M5),MX0=DM(I2,M1);
                    MY0=PM(I5,M5);                   {I5=I5+1}
                    DM(I0,M2)=MR0;
crosscorr_loop:
                    PM(I6,M6)=MR1;      { save result }
                 RTS;

.ENDMOD;
```

```
{----------------------------------}
{ Long Term Analysis Routine }
long_term_anal:
                AR=^gsm_byte+5;
                DM(gsm_byte_index)=AR;

                DM(d_ptr)=I0;                  { save I0 (d) }
                CNTR=4;
                DO long_term_loop UNTIL CE;
                    I1=DM(d_ptr);              { I1 = d }
                    I2=^wt;
                    CALL copy_residue_to_wt;
                    I1=^wt;
                    CALL dynamic_scaling2;
                    I1=`wt;
                    CALL long_term_param;     { gain and lag is now filled }

                    I1=DM(d_ptr);             { I1 --> d[0] }
                    AX1=^prev+3*SUBWIN;
                    AY1=DM(lag);
                    AR=AX1-AY1;
                    I5=AR;                    { AR = &prev[3*SUBWIN - lag] }
                    MX0=DM(gain);
                    CNTR=SUBWIN;
                    MY1=32768;
                    DO remove_predictable UNTIL CE;
                        MX1=DM(I1,M0);
                        MR=MX1*MY1(SU), MY0=PM(I5,M5);
                        { MY0 = prev[3*SUBWIN - lag +i }
                        MR=MR-MX0*MY0(SS);
remove_predictable:     DM(I1,M1)=MR1;       { d[i] = ... }
                    {
                        input:  VAR/DM  DM(d_ptr)      0..39
                        output: VAR/DM  x[40]          0..39
                    }
                    I0=DM(d_ptr);                          /* input */
                    I1=^x;                                 /* output */
                    CALL weighting_filter;
                    {
                        input:  VAR/DM x[40]       0..39    -> residue input
                        output: VAR/PM xM[13]      0..12    -> down sampling
residue
                                VAR/DM Mc;                  -> grid position
                    }
                    I0=^x;
                    I4=^xM;
                    CALL rpe_grid_selection;

                    {
                        input:  VAR/PM xM[13]              -> down sampled residue
                        output: VAR/DM xMc[13]             -> coded residue
                                VAR/DM mant                -> mantissa
                                VAR/DM expp                -> exponent
```

```
                    VAR/DM xmaxc           -> coded max residue
amplitude
              }
              I4=^xM;
              I0=^xMc;
              CALL APCM_quantization;
              {
                  input:  VAR/DM lag;             -> LTP lag
                          VAR/DM bc_out;          -> coded LTP gain
                          VAR/DM Mc;              -> RPE grid position
                          VAR/DM xmaxc;           -> coded max amplitude
                          VAR/DM xMc[13];         -> normalized RPE
sampling
                  output: VAR/PM gsm_byte[33] -> filled
              }
              I0=^xMc;
              I4=DM(gsm_byte_index);
              CALL encode_coded_rpe;


              I0=^xMc;
              I4=DM(gsm_byte_index);
              CALL decode_coded_rpe;

              AX0=DM(gsm_byte_index);
              AY0=7;
              AR=AX0+AY0;
              DM(gsm_byte_index)=AR;

              {
                  input:  VAR/DM xMc[13]        -> coded residue
                          VAR/DM mant           -> mantissa
                          VAR/DM expp           -> exponent
                  output: VAR/PM xMp[13]        -> down sampled residue
              }
              CALL APCM_inverse_quantization;

              {
                  input:  VAR/DM  Mc            -> grid position input
                          VAR/PM  xMp[13]       -> down sampled residue
                  output: VAR/DM  DM(d_ptr);    -> residue output
              }
              I0=DM(d_ptr);
              I4=^xMp;
              CALL rpe_grid_position;
              {
                  input:  VAR/DM  bc_out;
                  output: VAR/DM  gain;
              }
              CALL bc_out_to_gain;

              I1=DM(d_ptr);              { I1 --> d[0] }
              AX1=^prev+3*SUBWIN;
              AY1=DM(lag);
              AR=AX1-AY1;
              I5=AR;                     { AR = 3*SUBWIN - lag }
```

```
                    MX0=DM(gain);             { MX0 = gain }
                    CNTR=SUBWIN;
                    MY1=32768;
                    DO estimate_signal UNTIL CE;
                        MX1=DM(I1,M0);
                        MR=MX1*MY1(SU),MY0=PM(I5,M5);
                        { MY0 = prev[3*SUBWIN - lag +i }
                        MR=MR+MX0*MY0(SS);
estimate_signal:        DM(I1,M1)=MR1;        { d[i] = ...}
                    { shift the SUBWIN new estimates into the past. }
                    I4=^prev;
                    I5=^prev+SUBWIN;
                    CNTR=SUBWIN*2;
                    DO shift_2subwin UNTIL CE;
                          AR=PM(I5,M5);
shift_2subwin:   PM(I4,M5)=AR;

                    I1=DM(d_ptr);
                    CNTR=SUBWIN;
                    DO shift_1subwin UNTIL CE;
                          AR=DM(I1,M1);
shift_1subwin:   PM(I4,M5)=AR;

                    AX0=DM(d_ptr);
                    AY0=SUBWIN;
                    AR=AX0+AY0;
                    DM(d_ptr)=AR;             { I0 = d = d+SUBWIN }

long_term_loop:     NOP;
                RTS;


{---------------------------------}
long_term_param:
                SI=I1;                         { save I1: input residue }
                { setup parameters for calling crosscorrelation }
                                               { input: I1 = d }
                CALL crosscorrelate;           { 32 bit calculation for CC }
                I1=SI;                         { restore I1; }

                { find the maximum correlation }
                I6=^cc;                        { I6 points to cc msw }
                I0=^ccl;                       { I0 points to cc lsw }
                I4=^cc+1;                      { I4 = lag = 1, (^cc identical to
0)}
                CNTR=SUBWIN*2-1;
                AX1=PM(I6,M5);
                AX0=DM(I0,M1);
                { initialize maxcc -> AX = cc[0]}
                AY1=PM(I6,M5);
                AY0=DM(I0,M1);
                DO find_maxcc_loop UNTIL CE;
                    AR=AX1-AY1;
                    IF GT JUMP not_replace;
                    IF LT JUMP do_replace;
                    AR=AX0-AY0;
```

```
                  IF GE JUMP not_replace;
do_replace:
                     I4=I6;             { if cc[i] > maxcc then replace maxcc }
                     AX0=AY0;
                        AX1=AY1;
not_replace:      AY1=PM(I6,M5);
find_maxcc_loop:AY0=DM(I0,M1);

              MR1=AX1;
              MR0=AX0;                      { save maxcc into MR }
              SE=DM(maxcc_scale);
              SR=ASHIFT MR1(HI);
              SR=SR OR LSHIFT MR0(LO);

              DM(maxccl)=SR0;               { save maxcc (lsw) }
              DM(maxcc)=SR1;                { save maxcc (msw) }
              AX0=I4;
              AY0=^cc-SUBWIN+1;
              AR=AX0-AY0;                   { AR = *lag_out = lag + SUBWIN }
              DM(lag)=AR;                   { save lag }

              { calculate gain }

              AX0=^prev+3*SUBWIN;
              AY0=AR;
              AR=AX0-AY0;                   { AR = ^prev - *lag_out }
              I5=AR;                           { I5 = ^prev - *lag_out }
              AY0=1;
              I0=^energy;
              CALL calculate_energy;       { cross correlation cal. }
              AR=DM(energy);
              AR=PASS AR;
              IF EQ JUMP return_1;
div_for_gain:
              MR1=DM(energy);
              MR0=DM(energyl);
              SE=EXP MR1(HI);
              SR=NORM MR1(HI);
              SR=SR OR NORM MR0(LO);
              DM(energy)=SR1;
              DM(energyl)=SR0;

              MR1=DM(maxcc);
              MR0=DM(maxccl);
              SR=NORM MR1(HI);
              SR=SR OR NORM MR0(LO);
              DM(maxcc)=SR1;
              DM(maxccl)=SR0;

              /* quantization of gain */

              AR=1;
              DM(bc_out)=AR;
              CNTR=4;
              I4=^gsm_DLB;               { gain level table }
              MY0=DM(maxcc);
```

```
                DO compute_gain_level UNTIL CE;
                    MX0=DM(bc_out);
                    MR=MX0*MY0(SS);
                    AX0=MR0;
                    AX1=MR1;
                    AY0=PM(I4,M5);
                    AY1=0;
                    AR=AX0-AY0;
                    AR=AX1-AY1;
                    IF GT JUMP increase_level;
                    IF LT JUMP found_match_level;
                    AR=AX0-AY0;
                    IF GT JUMP increase_level;
found_match_level:
                        AX0=DM(bc_out);
                        AY0=1;
                        AR=AX0-AY0;
                        DM(bc_out)=AR;
                        POP CNTR, POP LOOP, POP PC;
                        JUMP end_compute_gain_level;
increase_level:
                    AX0=DM(bc_out);
                    AY0=1;
                    AR=AX0+AY0;
                    DM(bc_out)=AR;
compute_gain_level:
                    NOP;
end_compute_gain_level:

{----------------------------------}
calculate_energy:
                MR=0;MX0=PM(I5,M4);
                MY0=PM(I5,M5);
                CNTR=SUBWIN;
                DO cal_energy_loop UNTIL CE;
                    MR=MR+MX0*MY0(SS),MX0=PM(I5,M4);
cal_energy_loop:
            MY0=PM(I5,M5);
                DM(energy)=MR1;
                DM(energy1)=MR0;
                RTS;
```

ประวัติผู้เขียน

นายมีลาภ เรืองรัตนวิชา เกิดวันที่ 18 เมษายน พ.ศ. 2511 ที่อำเภอท่าม่วง จังหวัดกาญจนบุรี สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมไฟฟ้าสื่อสาร ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2532 และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตร์มหาบัณฑิต ที่จุฬาลงกรณ์มหาวิทยาลัย เมื่อ พ.ศ. 2536 ปัจจุบันทำงานอยู่ที่บริษัทเซ็นจูรี่ เทเลคอม จำกัด เขตดอนเมือง กรุงเทพมหานคร