```
/*************************************************************************
 *
 *  File:   clipfunc.dsp -
 *
 *  Purpose:     State machine for CLIP
 *
 *  Author: K.Narayanan
 *
 *  Date:   30.1.01
 *
 *  Bugs:
 *
 *  Change Log:  <Date> <Author>
 *          <Changes>
 *
 *************************************************************************/

.module/ram      Clip_Routines;

/* ---------------------------------------------------------------------- */
/* Include Files */
/* ------------- */
#include   <../stdinc/system.h>;
#include   <../stdinc/led.h>;         /* LEDx */
#include   <../stdinc/phlsp1.h>;    /* vars shared bet modem & sp1 */
#include   <../stdinc/nwksp1.h>;    /* vars shared bet network & sp1 */
#include   <../stdinc/def2181.h>;
#include   <../stdinc/bbireg.h>;
#include   <../stdinc/globid.h>;
#include   <../stdinc/eep.h>;
#include   <../stdinc/spy.h>;

/* ---------------------------------------------------------------------- */
/* Routines provided entry */
/* ----------------------- */
/* Routines written by  N.DeviPrasad Mon Oct  6 15:37:53 IST 1997      */
.entry  GenerateDTMFDigit;
.entry  CLISchedular_;

/* ---------------------------------------------------------------------- */
/* External Functions */
/* ------------------ */

/* ---------------------------------------------------------------------- */
/* Local Constants */
/* --------------- */

#if 0
#define    TONE_WIDTH       800   /* Corresponds to 100 mSec */
#define    INTER_DGT_WIDTH        480   /* Corresponds to  60 mSec */
#endif

/*  Various states for the dtmf dialling */
#define    STATE_IDLE       0
#define    STATE_SEND_TONE        1
#define         STATE_SEND_SIL         2

#define    MAX_DTMF_KEYS          16
.external Compress_;
/* ---------------------------------------------------------------------- */
```

```
/* External Variables */
/* ------------------ */
.external txVoiceData;
.external testVar10_;

/* ------------------------------------------------------------------------ */
/* Global Variables */
/* --------------- */
.var/dm/ram      dtmfDialDgt_;
.global               dtmfDialDgt_;
.var/dm/ram           dtmfGenState;
.global          dtmfGenState;

/* ------------------------------------------------------------------------ */
/* Local Variables */
/* --------------- */
.var/dm/ram      phaseLow, phaseHi;
.var/dm/ram           sinLow, sinHi;
.var/dm/ram      advanceLow, advanceHi;

.var/dm/ram      dtmfTimerCnt;

#ifdef HS
.var/dm/ram      localDgtVal;
#endif /* #ifdef HS */

/* As per the Application Book page No 443
 * Advance Value = 8.192 (toneFreq desired in Hz) */
/* Advance Values for Various sets of frequency */
#define    ADV_Fl0    0x164D     /*  697 Hz  '1', '2', '3', 'A',    */
#define    ADV_Fl1    0x18A3     /*  770 Hz  '4', '5', '6', 'B',    */
#define    ADV_Fl2    0x1B43     /*  852 Hz  '7', '8', '9', 'C',    */
#define    ADV_Fl3    0x1E1C     /*  941 Hz  '*', '0', '#', 'D';    */

#define    ADV_Fh0    0x26B0     /* 1209 Hz  -^               */
#define    ADV_Fh1    0x2AC0     /* 1336 Hz  ------^               */
#define    ADV_Fh2    0x2F43     /* 1477 Hz  -----------^          */
#define    ADV_Fh3    0x3441     /* 1633 Hz  ----------------^*/

#if 0
#define    ADV_Fh2    0x2E4D     /* 1447 Hz  -----------^          */
#endif
#if 0
#define    ADV_Fh2    0x2F43     /* 1477 Hz  -----------^          */
#define    ADV_Fh3    0x0000     /* 1633 Hz  ----------------^*/
#endif

.var/dm/ram advValueTbl[MAX_DTMF_KEYS * 2];   /* lo first, hi later */
.init      advValueTbl:
                ADV_Fl3, ADV_Fh1,      /* ASCII_ZERO    */
                ADV_Fl0, ADV_Fh0,/* ASCII_ONE     */
                ADV_Fl0, ADV_Fh1,      /* ASCII_TWO     */
                ADV_Fl0, ADV_Fh2,      /* ASCII_THREE   */
                ADV_Fl1, ADV_Fh0,      /* ASCII_FOUR    */
                ADV_Fl1, ADV_Fh1,/* ASCII_FIVE    */
                ADV_Fl1, ADV_Fh2,      /* ASCII_SIX     */
                ADV_Fl2, ADV_Fh0,/* ASCII_SEVEN   */
                ADV_Fl2, ADV_Fh1,      /* ASCII_EIGHT   */
                ADV_Fl2, ADV_Fh2,/* ASCII_NINE    */
                ADV_Fl3, ADV_Fh0,      /* ASCII_STAR    */
```

```
                ADV_Fl3, ADV_Fh2,/* ASCII_HASH    */
                ADV_Fl0, ADV_Fh3,/* ASCII_A  */
                ADV_Fl1, ADV_Fh3,/* ASCII_B  */
                ADV_Fl2, ADV_Fh3,/* ASCII_C  */
                ADV_Fl3, ADV_Fh3;/* ASCII_D  */


/*-------------------------------------------------------------------------
 *  GenerateDtmfDgt --  Generates the Dtmf sample for a given key pressed
 *     Args:     Nothing
 *    Returns:   Nothing
 *    Reg Affected: ar,af,ay0,ay1,my1,mx1,mf,mr,sr,i1,m3,ax0
 *    Bugs:
 * ------------------------------------------------------------------------*/
/* Output samples is written in txVoiceData */
/* to prevent echo cancellation */
GenerateDTMFDigit:
      dis m_mode; l1 = 0;

!     SPY (0xA002);

      ar = dm (dtmfGenState);
      af = pass ar;               /* In the Idle State */
      if eq jump ChkForKeyL;

      ay1 = STATE_SEND_TONE;      /* Send Tone For first 50 mSec */
      af = ar xor ay1;
      if eq jump SendToneL;

      ay1 = STATE_SEND_SIL;       /* Send Silence for the next 50 mSec */
      af = ar xor ay1;
      if eq jump SendSilL;

      rts;

ChkForKeyL:
!     SPY (0xA003);

      sr0 = dm (testVar10_);
      af = pass sr0;
      if eq jump Default1L;

      dm (dtmfDialDgt_) = sr0;

Default1L:
      ar = dm (dtmfDialDgt_);           /* Check if there is any Dtmf Digit */
      af = pass ar;               /* to be processed.          */
      if eq rts;

#ifdef HS
      dm (localDgtVal) = ar;            /* store dtmfDialDgt in local var */
#else /* #ifdef HS ... else ... */
      sr0 = dm (testVar10_);
      af = pass sr0;
      if ne jump SkipDigitResetL;

      ay1 = 0; dm (dtmfDialDgt_) = ay1; /* Reset the Value */

SkipDigitResetL:
```

```
#endif /* #ifdef HS ... else ... */

      ay1 = ASCII_HASH;                 /* Ascii Hash */
      af = ar xor ay1;
      if eq jump HashPressedL;

      ay1 = ASCII_STAR;                 /* Ascii Star */
      af = ar xor ay1;
      if eq jump StarPressedL;

      ay1 = ASCII_A;                    /* Ascii A */
      af = ar xor ay1;
      if eq jump APressedL;

      ay1 = ASCII_B;                    /* Ascii B */
      af = ar xor ay1;
      if eq jump BPressedL;

      ay1 = ASCII_C;                    /* Ascii C */
      af = ar xor ay1;
      if eq jump CPressedL;

      ay1 = ASCII_D;                    /* Ascii D */
      af = ar xor ay1;
      if eq jump DPressedL;

      /* Some Other Key is pressed Chk if it is 0 to 9 */
      ay1 = ASCII_ZERO;
      ar = ar - ay1;                /* ar = keyPressed - 0x30 */
      if lt rts;              /* return if it is less than 0 */

      ay1 = 9;                /* Chk whether Dgt is B'ween 0 & 9 */
      af = ar - ay1;                /* Or # or *, Other Keys no actions */
      if gt rts;

      jump LoadParL;                /* DTMF_OFFSET in ar */

StarPressedL:
      ar = 10;                /* Load Offset as 10 for ASCII_STAR */
      jump LoadParL;                /* DTMF_OFFSET in ar */

HashPressedL:
      ar = 11;                /* Load Offsetr as 11 for ASCII_HASH */
      jump LoadParL;                /* DTMF_OFFSET in ar */

APressedL:
BPressedL:
CPressedL:
DPressedL:
      ar  = ASCII_A;
      ar  = ay1 - ar;
      ay1 = 12;
      ar  = ar + ay1;
      jump LoadParL;                /* DTMF_OFFSET in ar */

LoadParL:
      SPY (0xA004);
      SPY (ar);

      /* The Offset Val (in ar) will be b'ween 0 and 11 */
```

```
      /* Load the Freq, Phase and time duration and change the state */
      i1 = ^advValueTbl;
      sr = lshift ar by 1 (lo);
      m3 = sr0;
      modify (i1,m3);
      ar = dm (i1,m1); dm (advanceLow) = ar;  /* str two advance Values */
      SPY (sr0);
      SPY (ar);

      ar = dm (i1,m1); dm (advanceHi) = ar;
      SPY (ar);

/*    Reset the phase value initially */
      ar = 0; dm (phaseLow) = ar; dm (phaseHi) = ar;
      dm (txVoiceData) = ar;       /* sine(0) + sine(0) = 0 */

      ar = STATE_SEND_TONE;  dm (dtmfGenState) = ar;

      ar  = dm (clipFlag_);  /* this cnt is in multiple of 50 ms */
      ay1 = 0x07;
      ar  = ar and ay1;

      ena m_mode;
      mr0 = 320;       /* 000:40msec,111:110msec */
      my1 = 80;        /* convert to  125 us count  and load */
      mr = mr + ar * my1 (uu);     /* in internal counter */
      dis m_mode;

!     ar = TONE_WIDTH;
      dm (dtmfTimerCnt) = mr0;
      rts;

SendToneL:
/*    Advance the phase by the factor determined */
      ar = dm (advanceLow);
      ay1 = dm (phaseLow);
      ar = ar + ay1;
      dm (phaseLow) = ar;


/* Lower Freq should be at -8 dbm and Higher Freq is at -6 dbm .
Calculate its sine Value  , parameter in ax0. Output is shifted by 3 bits
to right to get a dbm value which corresponds to amplitude  level of
Codec .To reduce again -6 dbm value shift by -1 . So overall we shift
by -4 to right to get a dbm level of -6 . For a -2db factor is 0.63
which corresponds to 0x65AC in hex */
      ax0 = ar;
      call sin;        /* Ret Value in ar */

#if 1
      sr = ashift ar by -4 (hi);
!     sr = ashift ar by -6 (hi);
      my1 = 0x65AC;                /* -8 dbm */
      mr = sr1 * my1 (ss);
      dm (sinLow) = mr1;
#else   /* trial an error for Brazil */
      sr = ashift ar by -3 (hi);
      dm (sinLow) = sr1;
#endif
```

```
/*    Advance the value by the factor determined */
      ar = dm (advanceHi);
      ay1 = dm (phaseHi);
      ar = ar + ay1;
      dm (phaseHi) = ar;

      ax0 = ar;
      call sin;         /* Ret value in ar */
!     sr = ashift ar by -6 (hi);
      sr = ashift ar by -4 (hi);/* trial an error for Brazil */
      dm (sinHi) = sr1;

AddBothFreq:
      ar = dm (sinLow);
      ay1 = dm  (sinHi);
      ar = ar + ay1;
      call Compress_;
      dm (txVoiceData) = ar;       /* OutPut Sample */

#ifdef HS
      ar = dm (localDgtVal);
      ay1 = dm (dtmfDialDgt_);
      af = ar xor ay1;
      if eq rts;
#else /* #ifdef HS ... else ... */
      ar = dm (dtmfTimerCnt);
      ar = ar - 1;
      dm (dtmfTimerCnt) = ar;
      if ne rts;
#endif /* #ifdef HS ... else ... */

ChangeToSilenL:
      /* Reload timer width for Silence and change state */
      ar  = dm (clipFlag_);  /* this cnt is in multiple of 50 ms */
      sr  = lshift ar by -3 (lo);
      ay1 = 0x07;
      ar  = sr0 and ay1;

      ena m_mode;
      mr0 = 320;        /* 000:40msec,111:110msec */
      my1 = 80;         /* convert to  125 us count  and load */
      mr = mr + ar * my1 (uu);     /* in internal counter */
      dis m_mode;

!     ar = INTER_DGT_WIDTH;
      dm (dtmfTimerCnt) = mr0;

      ar = STATE_SEND_SIL; dm (dtmfGenState) = ar;

      SPY (0xA007);
      rts;


SendSilL:         /* In the State Silence */
      /* Voice is Sent But the Silence State is maintained */
!     ar = 0; dm (txVoiceData) = ar;

      ar = dm (dtmfTimerCnt);
      ar = ar - 1;
      dm (dtmfTimerCnt) = ar;
```

```
      if ne rts;

ChangeToIdleL:
      /* go back to looking for the next dtmf digit */
      SPY (0xA008);
      ar = STATE_IDLE; dm (dtmfGenState) = ar;
      rts;

/*-------------------------------------------------------------------------
 *  sin --  Sine Code that has been taken from R.Balaji
 *     Args:
 *    Returns:   Nothing
 *    Bugs:
 * ------------------------------------------------------------------------*/
/*
        Sine Approximation
                Y = Sin(x)

        Calling Parameters
                AX0 = x in scaled 1.15 format
           m1 = 1;l1 = 0;
        Return Values
                AR = y in 1.15 format

        Altered Registers
                AY0,AF,AR,MY1,MX1,MF,MR,SR,I1

        Computation Time
                25 cycles
*/

.var/dm/ram      sin_coeff[5];
.init       sin_coeff:  0x3240,0x0053,0xaacc,0x08b7,0x1cce;

sin:    i1 = ^sin_coeff;                {Pointer to coeff. buffer}
        ay0 = h#4000;
        ar = ax0, af = ax0 and ay0;           {Check 2nd or 4th quad.}
        if ne ar = -ax0;                {If yes, negate input}
        ay0 = h#7fff;
        ar = ar and ay0;                {Remove sign bit}
        my1 = ar;
        mf = ar*my1 (rnd), mx1 = dm(i1,m1);   {mf = x2}
        mr = mx1*my1 (ss), mx1 = dm(i1,m1);   {mr = C1x}
        cntr = 3;
        do Approx until ce;
                mr = mr+mx1*mf (ss);
Approx:         mf = ar*mf (rnd), mx1 = dm(i1,m1);
        mr = mr+mx1*mf (ss);
        sr = ashift mr1 by 3 (hi);
        sr = sr or lshift mr0 by 3 (lo);      {Convert to 1.15 format}
        ar = pass sr1;
        if lt ar = pass ay0;            {Saturate if needed}
        af = pass ax0;
        if lt ar = -ar;                 {Negate output if needed}
        rts;

/*-------------------------------------------------------------------------
 *  CLISchedular_ --  State Machine for CLIP
 *     Args:     Nothing
 *    Returns:   Nothing
```

```
 *    Reg Affected:
 *    Bugs:
 * ------------------------------------------------------------------------*/
CLISchedular_:
.var/dm/ram prevCliState_;
.var/dm/ram intCliTmrCnt_;
.var/dm/ram cliDigitPtr;
#if 0
.var/dm/ram cliDigitBuf[10];
.init cliDigitBuf:0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,CLI_LAST_DIGIT;
#endif
      l1 = 0;     l6 = 0;m1 = 1;

      ar = dm(cliPgmState_);
      af = pass ar;
      if ge jump NonNegativeStateL;

      ar = 0x5C01; jump Freeze_;

NonNegativeStateL:
      ay1 = MAX_CLI_STATES;
      af = ar - ay1;
      if lt jump CLIContinueL;
      ar = 0x5C00; jump Freeze_;   /* invalid cliPgmState */

CLIContinueL:
      i6 = ^CLIJumpTable;
      m5 = ar;
      modify (i6,m5);
      jump (i6);

CLIJumpTable:
      jump CLIIdleL;
      jump CLIStart1L;
      jump CLIStart2L;
      jump CLIDTMFDgtWaitL;
      jump CLIBegGuardL;
      jump CLISendDtmfDgtL;
      jump CLIEndL;
      jump CLIEndGuardL;
      jump CLIWaitStart1L;

CLIIdleL:
      rts;

CLIStart1L:
      sr0 = dm(clipStart1_);
      ay1 = 0x1f;
      ar = sr0 and ay1;/* this cnt is in multiple of 50 ms */
      my1 = 400;       /* convert to  125 us count  and load */
      mr = ar * my1 (uu);    /* in internal counter */
      dm (intCliTmrCnt_) = mr0;

CheckRevL:
      af = tstbit 7 of sr0;
      if eq jump SkipRevL;
      ar = BATT_REVERSE;dm (linePolarity_) =ar ; /* Give Battery Reversal */
SkipRevL:
      ar = CLI_WAIT_START1; dm( cliPgmState_) = ar;
      rts;
```

```
CLIWaitStart1L:
      ar = dm (intCliTmrCnt_);
      af = pass ar;
      if eq jump ChangeStateBegGuardL;
      ar = ar - 1;
      dm (intCliTmrCnt_) = ar;
      if ne rts;

ChangeStateBegGuardL:
      ar  = CLI_BEG_GUARD; dm ( cliPgmState_) = ar;

      mr0 = dm (beginGuard_);      /* this cnt is in multiple of 50 ms */
      my1 = 400;        /* convert to  125 us count  and load */
      mr = mr0 * my1 (uu);   /* in internal counter */
      dm (intCliTmrCnt_) = mr0;

      rts;

CLIBegGuardL:
      ar = dm (intCliTmrCnt_);
      af = pass ar;
      if eq jump ChangeStart2L;
      ar = ar - 1;
      dm (intCliTmrCnt_) = ar;
      if ne rts;

ChangeStart2L:
      SPY(0x5d0A);
      ar = CLI_START2; dm( cliPgmState_) = ar;
      rts;



CLIStart2L:
      SPY(0x5d01);
      ar = dm (clipStart2_);
      af = pass ar;
      if eq jump ChangeStateDgtL;  /* no spl DTMF start dgt */

      dm (dtmfDialDgt_) = ar;
      ar = dm( cliPgmState_); dm(prevCliState_) = ar;
      ar = CLI_DTMF_DGT_WAIT; dm( cliPgmState_) = ar;
      rts;

CLISendDtmfDgtL:
      SPY(0x5d03);
      i1 = dm (cliDigitPtr);
      ar  = dm (i1,m1);
      dm (cliDigitPtr) = i1;
      ay1 = CLI_LAST_DIGIT;
      af  = ar xor ay1;
      if eq jump ChangeStateCliEndL;

      dm (dtmfDialDgt_) = ar;
      ar = dm( cliPgmState_); dm(prevCliState_) = ar;
      ar = CLI_DTMF_DGT_WAIT; dm( cliPgmState_) = ar;
      rts;
```

```
ChangeStateCliEndL:
      ar = CLI_END; dm ( cliPgmState_) = ar;
      rts;

CLIEndL:
      SPY(0x5d04);
      ar = dm (clipEnd_);
      af = pass ar;
      if eq jump ChangeStateEndGuardL;  /* no spl DTMF start dgt */

      dm (dtmfDialDgt_) = ar;
      ar = dm( cliPgmState_); dm(prevCliState_) = ar;
      ar = CLI_DTMF_DGT_WAIT; dm( cliPgmState_) = ar;
      rts;

CLIEndGuardL:
      ar = dm (intCliTmrCnt_);
      af = pass ar;
      if eq jump ChangeStateIdleL;
      ar = ar - 1;
      dm (intCliTmrCnt_) = ar;
      if ne rts;
ChangeStateIdleL:
      SPY(0x5d05);

      ar = CLI_IDLE; dm( cliPgmState_) = ar;
      rts;

CLIDTMFDgtWaitL:
      ar = dm (dtmfGenState);
      ay1 = STATE_IDLE;
      af = ar xor ay1;
      if ne rts;

      ar  = dm (prevCliState_);
      ay1 = CLI_START2;
      af  = ar xor ay1;
      if eq jump ChangeStateDgtL;

      ay1 = CLI_SEND_DTMF_DGT;
      af  = ar xor ay1;
      if eq jump ChangeStateSndDtmfDgtL;

      ay1 = CLI_END;
      af  = ar xor ay1;
      if eq jump ChangeStateEndGuardL;

      ar = 0x5C02;     jump Freeze_;    /* Invalid prevCliState */

ChangeStateDgtL:
      SPY(0x5d02);
      ar = CLI_SEND_DTMF_DGT; dm( cliPgmState_) = ar;

      i1 = ^cliDigitBuf_;
      dm (cliDigitPtr) = i1;
      rts;

ChangeStateEndGuardL:
      ar = CLI_END_GUARD; dm ( cliPgmState_) = ar;
```

```
      mr0 = dm (endGuard_);  /* this cnt is in multiple of 50 ms */
      my1 = 400;       /* convert to  125 us count  and load */
      mr = mr0 * my1 (uu);   /* in internal counter */
      dm (intCliTmrCnt_) = mr0;
      rts;

ChangeStateSndDtmfDgtL:
      ar = CLI_SEND_DTMF_DGT; dm ( cliPgmState_) = ar;
      rts;


/*-----------------------------    End of File  -------------------------*/
.endmod;
/*-----------------------------    End of File  -------------------------*/
```