CODE: Low Noise Cancellation

```
#include "sinecfg.h"
#include <math.h>

// ---------------------------------------------------------------------------
// Defines
// ---------------------------------------------------------------------------

// Common data types
#define UINT   unsigned int
#define UINT16 unsigned int
#define INT16  signed   int
#define UINT32 unsigned long
#define INT32  signed   long
#define BOOL   unsigned short

// EMIF registers
#define EGCR            0x0800
#define EMI_RST         0x0801
#define EMI_BE          0x0802
#define CE01            0x0803
#define CE02            0x0804
#define CE03            0x0805
#define CE11            0x0806
#define CE12            0x0807
#define CE13            0x0808
#define CE21            0x0809
#define CE22            0x080A
#define CE23            0x080B
#define CE31            0x080C
#define CE32            0x080D
#define CE33            0x080E
#define SDC1            0x080F
#define SDPER           0x0810
#define INIT            0x0812
#define SDC2            0x0813
#define SDC3            0x0814

// MCBSP registers
#define DRR2_0          0x2800
#define DRR1_0          0x2801
#define DXR2_0          0x2802
#define DXR1_0          0x2803
#define SPCR2_0         0x2804
#define   XRST          0
#define   XRDY          1
#define   GRST          6
#define SPCR1_0         0x2805
#define   RRST          0
#define   RRDY          1
#define RCR2_0          0x2806
#define RCR1_0          0x2807
#define XCR2_0          0x2808
#define XCR1_0          0x2809
#define SRGR2_0         0x280A
#define SRGR1_0         0x280B
#define MCR2_0          0x280C
#define MCR1_0          0x280D
#define RCERA_0         0x280E
#define RCERB_0         0x280F
#define XCERA_0         0x2810
#define XCERB_0         0x2811
```

```
#define PCR0            0x2812
#define RCERC_0         0x2813
#define RCERD_0         0x2814
#define XCERC_0         0x2815
#define XCERD_0         0x2816
#define RCERE_0         0x2817
#define RCERF_0         0x2818
#define XCERE_0         0x2819
#define XCERF_0         0x281A
#define RCERG_0         0x281B
#define RCERH_0         0x281C
#define XCERG_0         0x281D
#define XCERH_0         0x281E

// I2C registers
#define I2COAR                  0x3C00
#define I2CIER                  0x3C01
#define I2CSTR                  0x3C02
#define   NACK          1
#define   I2C_XRDY      4
#define   BB            12
#define I2CCLKL                 0x3C03
#define I2CCLKH                 0x3C04
#define I2CCNT                  0x3C05
#define I2CDRR                  0x3C06
#define I2CSAR                  0x3C07
#define I2CDXR                  0x3C08
#define I2CMDR                  0x3C09
#define   RM            7
#define   TRX           9
#define   MST           10
#define   STP           11
#define   STT           13
#define I2CISRC                 0x3C0A
#define I2CGPIO                 0x3C0B
#define I2CPSC                  0x3C0C

// Memory-mapped registers
#define USER_REG        0x3F0000
#define DC_REG          0x3F0001
#define MISC_REG        0x3F0006

// AIC23 I2C address and I2C registers
#define AIC23_I2CADDR         0x1a
#define AIC23_LEFTINVOL       0
#define AIC23_RIGHTINVOL      1
#define AIC23_LEFTHPVOL       2
#define AIC23_RIGHTHPVOL      3
#define AIC23_ANAPATH         4
#define AIC23_DIGPATH         5
#define AIC23_POWERDOWN       6
#define AIC23_DIGIF           7
#define AIC23_SAMPLERATE      8
#define AIC23_DIGACT          9
#define AIC23_RESET           15

// ---------------------------------------------------------------------------
// Pre-generated sine wave data, 16-bit signed samples
// Value is 32767 * sin(n / 48 * 2 * PI); 0 <= n <= 47
// ---------------------------------------------------------------------------

#define FREQ_RANGE 11
#define LUT_SIZE   256
```

```
int lut[LUT_SIZE] = {
            0x0000,0x0324,0x0648,0x096A,0x0C8C,0x0FAB,0x12C8,0x15E2,
            0x18F9,0x1C0B,0x1F1A,0x2223,0x2528,0x2826,0x2B1F,0x2E11,
            0x30FB,0x33DF,0x36BA,0x398C,0x3C56,0x3F17,0x41CE,0x447A,
            0x471C,0x49B4,0x4C3F,0x4EBF,0x5133,0x539B,0x55F5,0x5842,
            0x5A82,0x5CB3,0x5ED7,0x60EB,0x62F1,0x64E8,0x66CF,0x68A6,
            0x6A6D,0x6C23,0x6DC9,0x6F5E,0x70E2,0x7254,0x73B5,0x7504,
            0x7641,0x776B,0x7884,0x7989,0x7A7C,0x7B5C,0x7C29,0x7CE3,
            0x7D89,0x7E1D,0x7E9C,0x7F09,0x7F61,0x7FA6,0x7FD8,0x7FF5,
            0x7FFF,0x7FF5,0x7FD8,0x7FA6,0x7F61,0x7F09,0x7E9C,0x7E1D,
            0x7D89,0x7CE3,0x7C29,0x7B5C,0x7A7C,0x7989,0x7884,0x776B,
            0x7641,0x7504,0x73B5,0x7254,0x70E2,0x6F5E,0x6DC9,0x6C23,
            0x6A6D,0x68A6,0x66CF,0x64E8,0x62F1,0x60EB,0x5ED7,0x5CB3,
            0x5A82,0x5842,0x55F5,0x539B,0x5133,0x4EBF,0x4C3F,0x49B4,
            0x471C,0x447A,0x41CE,0x3F17,0x3C56,0x398C,0x36BA,0x33DF,
            0x30FB,0x2E11,0x2B1F,0x2826,0x2528,0x2223,0x1F1A,0x1C0B,
            0x18F9,0x15E2,0x12C8,0x0FAB,0x0C8C,0x096A,0x0648,0x0324,
            0x0000,0xFCDC,0xF9B8,0xF696,0xF374,0xF055,0xED38,0xEA1E,
            0xE707,0xE3F5,0xE0E6,0xDDDD,0xDAD8,0xD7DA,0xD4E1,0xD1EF,
            0xCF05,0xCC21,0xC946,0xC674,0xC3AA,0xC0E9,0xBE32,0xBB86,
            0xB8E4,0xB64C,0xB3C1,0xB141,0xAECD,0xAC65,0xAA0B,0xA7BE,
            0xA57E,0xA34D,0xA129,0x9F15,0x9D0F,0x9B18,0x9931,0x975A,
            0x9593,0x93DD,0x9237,0x90A2,0x8F1E,0x8DAC,0x8C4B,0x8AFC,
            0x89BF,0x8895,0x877C,0x8677,0x8584,0x84A4,0x83D7,0x831D,
            0x8277,0x81E3,0x8164,0x80F7,0x809F,0x805A,0x8028,0x800B,
            0x8001,0x800B,0x8028,0x805A,0x809F,0x80F7,0x8164,0x81E3,
            0x8277,0x831D,0x83D7,0x84A4,0x8584,0x8677,0x877C,0x8895,
            0x89BF,0x8AFC,0x8C4B,0x8DAC,0x8F1E,0x90A2,0x9237,0x93DD,
            0x9593,0x975A,0x9931,0x9B18,0x9D0F,0x9F15,0xA129,0xA34D,
            0xA57E,0xA7BE,0xAA0B,0xAC65,0xAECD,0xB141,0xB3C1,0xB64C,
            0xB8E4,0xBB86,0xBE32,0xC0E9,0xC3AA,0xC674,0xC946,0xCC21,
            0xCF05,0xD1EF,0xD4E1,0xD7DA,0xDAD8,0xDDDD,0xE0E6,0xE3F5,
            0xE707,0xEA1E,0xED38,0xF055,0xF374,0xF696,0xF9B8,0xFCDC
};


//delta phases pre caluclated by using the formula (delta_phase = f/fs* 256) where 40<=f
<=230
//fs = 48000 Hz and 256 phases on the unit circle
//The delta phases are decimal numbers and hence have been scaled by 2^16 to represent in
the Q16.16 format
//On scaling they are rounded to real numbers and their hex values are obtained
//These hex values form the entries of the delta_pase[20] array defined below
//long data type is 32 bits wide and thus used for Q16.16 format
//Shifting these hex values to right by 16 bits will give the desired phase index for the
look up table(lut)

long unsigned delta_phase[FREQ_RANGE] = {0x00001B4F,
            0x000028F6,
            0x0000369D,
            0x00004444,
            0x000051EC,
            0x00005F93,
            0x00006D3A,
            0x00007AE1,
            0x00008889,
            0x00009630,
            0x0000A3D7
};

int cycles[FREQ_RANGE] = {20,30,40,50,60,70,80,90,100,110,120};
int freq[FREQ_RANGE];
long unsigned energy[FREQ_RANGE];
```

```
long long unsigned a0[FREQ_RANGE] = {16771631,
            16771631,
            16771631,
            16771631,
            16771631,
            16771631,
            16763259,
            16771631,
            16771631,
            16771631,
            16771631


 };
long long unsigned a1[FREQ_RANGE] = {16765930,
            16765786,
            16765584,
            16765326,
            16765011,
            16764636,
            16747463,
            16763717,
            16763172,
            16762568,
            16761907



};
long long unsigned a2[FREQ_RANGE] = {16771631,
            16771631,
            16771631,
            16771631,
            16771631,
            16771631,
            16763259,
            16771631,
            16771631,
            16771631,
            16771631


};
long long unsigned b1[FREQ_RANGE] = {16765930,
            16765786,
            16765584,
            16765326,
            16765011,
            16764636,
            16747463,
            16763717,
            16763172,
            16762568,
            16761907



};
```

```
long long unsigned b2[FREQ_RANGE] = {16766044,
            16766044,
            16766044,
            16766044,
            16766044,
            16766044,
            16749302,
            16766044,
            16766044,
            16766044,
            16766044



};

int yl[11],xl[11],yr[3],xr[3];
long long unsigned a[11] = {122,
            1219,
            5484,
            14624,
            25592,
            30710,
            25592,
            14624,
            5484,
            1219,
            122
};
long long unsigned b[11] = {1,
            19924,
            30195,
            28185,
            20387,
            10545,
            4144,
            1157,
            225,
            27,
            1
};


// ----------------------------------------------------------------------------
// Helper functions
// ----------------------------------------------------------------------------

void outportw(UINT port, UINT value)
{
  ioport UINT *pptr;
  pptr = (ioport UINT*) port;
  *pptr = value;
}

void outportf(UINT port, UINT bit, UINT value)
{
  ioport UINT *pptr;
  pptr = (ioport UINT*) port;
  if (value != 0)
    *pptr |= (1 << bit);
  else
    *pptr &= ~(1 << bit);
}
```

```
UINT inportw(UINT port)
{
  ioport UINT *pptr;
  pptr = (ioport UINT*) port;
  return *pptr;
}

UINT inportf(UINT port, UINT bit)
{
  ioport UINT *pptr;
  pptr = (ioport UINT*) port;
  return ((*pptr & (1 << bit)) != 0);
}

void outmemw(UINT32 add, UINT value)
{
  UINT *mptr;
  mptr = (UINT*) add;
  *mptr = value;
}

void outmemf(UINT32 add, UINT bit, UINT value)
{
  UINT *mptr;
  mptr = (UINT*) add;
  if (value != 0)
    *mptr |= (1 << bit);
  else
    *mptr &= ~(1 << bit);
}

UINT inmemw(UINT32 add)
{
  UINT *mptr;
  mptr = (UINT*) add;
  return *mptr;
}

UINT inmemf(UINT32 add, UINT bit)
{
  UINT *mptr;
  mptr = (UINT*) add;
  return ((*mptr & (1 << bit)) != 0);
}

// ---------------------------------------------------------------------------
// I2C functions
// ---------------------------------------------------------------------------

int i2c_write(UINT *data,int length, int address)
{
  int ok = 1;
  int time;
  int timeout = 1000;
  int count, i;

  // Enable tx mode
  outportf(I2CMDR, TRX, 1);

  // Set data write count so device will know when to send STOP bit
  outportw(I2CCNT, length);
```

```
  // Set slave address
  outportw(I2CSAR, address);

  // Master mode
  outportf(I2CMDR, MST, 1);

  // Set mode to use I2CCNT), send stop when count zero
  outportf(I2CMDR, RM, 0);
  outportf(I2CMDR, STP, 1);

  // Check if bus busy
  time = 0;
  while ((time++ < timeout) && (inportf(I2CSTR, BB) != 0))
    asm (" NOP");
  ok = (time < timeout);

  // Main loop
  if (ok)
  {
    count = 0;
    while ((count < length) && ok)
    {
      // Check if transmitter ready
      time = 0;
      while ((time++ < timeout) && (inportf(I2CSTR, I2C_XRDY) == 0))
        asm (" NOP");
      ok = (time < timeout);
      if (ok)
      {
        // Write data byte into transmit buffer
        outportw(I2CDXR, *data++);

        // Set start condition after first byte written
            if (count == 0)
              outportf(I2CMDR, STT, 1);

        // Wait for ack/nack
        for (i = 0; i < 32000; i++)
          asm (" NOP");

        // Check if nack
        time = 0;
        while ((time++ < timeout) && (inportf(I2CSTR, NACK) != 0))
          asm (" NOP");
        count++;
      }
      ok = (time < timeout);
    }
  }
  return ok;
}

void i2c_init()
{
  outportw(I2CMDR, 0x0620);
  outportw(I2CPSC, 0x0006);
  outportw(I2CCLKL, 0x000F);
  outportw(I2CCLKH, 0x000F);
}

// ---------------------------------------------------------------------------
//  Codec functions
// ---------------------------------------------------------------------------
```

```
void aic23_write_cmd(int regnum, int regval)
{
    UINT buf[2];

    buf[0] = (UINT)((regnum<<1)+(regval>>8));
    buf[1] = (UINT)(regval & 0x00FF);

    i2c_write(buf, 2, AIC23_I2CADDR);
}

void aic23_open()
{
  // Reset the AIC23
  aic23_write_cmd(AIC23_RESET, 0);

  // Turn everything on in AIC23
  aic23_write_cmd(AIC23_POWERDOWN, 0);

  // Set input gain to 0dB, mute off
  aic23_write_cmd(AIC23_LEFTINVOL, 0x0017);
  aic23_write_cmd(AIC23_RIGHTINVOL, 0x0017);

  // Set headphone gain to -40 dB, minimize clicks
  // NOTE: Use caution when changing these values to prevent ear damage!!
  aic23_write_cmd(AIC23_LEFTHPVOL, 0x00EE); //0x00D1
  aic23_write_cmd(AIC23_RIGHTHPVOL, 0x00EE); //0x00D1

  // Enable DAC, use mic with 20 dB gain
  aic23_write_cmd(AIC23_ANAPATH, 0x0015);

  // No de-emphasis filter, ADC HPF enabled
  aic23_write_cmd(AIC23_DIGPATH, 0x0000);

  // Master mode, sync followed by 2 16-bit samples format
  aic23_write_cmd(AIC23_DIGIF, 0x0043);

  // Enable 48 ksps rate with 12 MHz MCLK
  aic23_write_cmd(AIC23_SAMPLERATE, 0x0081);

  // Activate interface
  aic23_write_cmd(AIC23_DIGACT, 0x0001);
}

BOOL aic23_write_data(int value)
{
  BOOL result;
  // Wait until mcbsp channel 0 can accept data
  result = (inportf(SPCR2_0, XRDY) != 0);
  // Write data to mcbsp channel 0 if ready
  if (result)
      outportw(DXR1_0, value);
  return result;
}

BOOL aic23_read_data(int *value)
{
  BOOL result;
  // Check if mcbsp channel 0 can accept data
  result = (inportf(SPCR1_0, RRDY) != 0);
  // Write data to mcbsp channel 0 if ready
  if (result)
      *value = inportw(DRR1_0);
  return result;
```

```
}


BOOL aic23_w_data(long signed value)
{
  BOOL result;
  // Wait until mcbsp channel 0 can accept data
  result = (inportf(SPCR2_0, XRDY) != 0);
  // Write data to mcbsp channel 0 if ready
  if (result)
      outportw(DXR1_0, value);
  return result;
}

BOOL aic23_r_data(long signed *value)
{
  BOOL result;
  // Check if mcbsp channel 0 can accept data
  result = (inportf(SPCR1_0, RRDY) != 0);
  // Write data to mcbsp channel 0 if ready
  if (result)
      *value = inportw(DRR1_0);
  return result;
}



void aic23_close()
{
  // Put codec in slave mode to stop driving outputs
  aic23_write_cmd(AIC23_DIGIF, 0x0003);

  // Turn the codec off
  aic23_write_cmd(AIC23_POWERDOWN, 0x00ff);
}

// ------------------------------------------------------------------------
// Hardware initialization
// ------------------------------------------------------------------------

void mcbsp_init()
{
  int i;

  // Setup control registers
  outportw(SPCR1_0, 0x0000);
  outportw(SPCR2_0, 0x0000);

  // Receive frame format is single phase, two 16-bit words
  outportw(RCR1_0, 0x0140);
  outportw(RCR2_0, 0x0000);

  // Transmit frame format is single phase, two 16-bit words
  outportw(XCR1_0, 0x0140);
  outportw(XCR2_0, 0x0000);

  // Use external clock and sync for TX and RX
  outportw(PCR0, 0x0003);

  // Sample clock generator don't care, no multichannel setting
  // Disable RCERx_0 and XCERx_0 registers
  outportw(SRGR1_0, 0x0000);
  outportw(SRGR2_0, 0x0000);
```

```
  outportw(MCR1_0, 0x0000);
  outportw(MCR2_0, 0x0000);
  for (i = 0; i < 4; i++)
    outportw(RCERA_0 + i, 0x0000);
  for (i = 0; i < 12; i++)
    outportw(RCERC_0 + i, 0x0000);

  // Empty mcbsp channel 0 receive buffer
  while (inportf(SPCR1_0, RRDY) != 0)
    inportw(DRR1_0);

  // Reset sample rate
  outportf(SPCR2_0, GRST, 1);

  // Reset receiver
  outportf(SPCR1_0, RRST, 1);

  // Reset transmitter
  outportf(SPCR2_0, XRST, 1);
}

void hardware_init()
{
  // Initialize I2C
  i2c_init();

  // Initialize MCBSP
  mcbsp_init();

  // Clear CPLD registers (USER_REG, DC_REG, and MISC)
  outmemw(USER_REG, 0x0000);
  outmemw(DC_REG, 0x0000);
  outmemw(MISC_REG, 0x0000);
}



void test_MIC()
{
signed int temp;
      while(1)
        {
            while(!aic23_read_data(&temp));
            while(!aic23_write_data(temp));
            while(!aic23_read_data(&temp));
            while(!aic23_write_data(temp));
        }

}



void sort()
{
int i = 0, j = 0;
long unsigned temp, temp1;

for(i = 0; i< FREQ_RANGE; i++)
      for(j = i+1 ; j< FREQ_RANGE; j++)
      {
            if(energy[j] > energy[i])
            {
                  temp = energy[i];
```

```
                  temp1 = energy[j];
                  energy[i] = energy[j];
                  energy[j] = temp;
                  temp = freq[i];
                  freq[i] = freq[j];
                  freq[j] = temp;

            }


      }

}


void test_LED()
{
      unsigned int i, k;
      unsigned long j;
      while(1)
      for(i = 0;i < 16;i++){
            outmemw(USER_REG, i);
            for(j = 0;j<5000000;j++);
        }


}


void test_LineIn()
{
      int temp, temp1;
      //int r1,r2,r3,r4,r5,r;
      long long r,result1, result[3];

      aic23_write_cmd(AIC23_ANAPATH, 0x0010);
      while (inmemf(USER_REG, 4) != 0);

      while(1)
      {
               while (!aic23_read_data(&temp));
               if(inmemf(USER_REG, 5) == 0){
                     while(!aic23_write_data(temp));
                     while (!aic23_read_data(&temp1));
                     while(!aic23_write_data(temp1));

               }


                           xl[10] = temp;

           result[0] =  (b[9]*yl[1])+ (b[7]*yl[3])+ (b[5]*yl[5])+
(b[3]*yl[7])+(b[1]*yl[9])+(a[10]*xl[0])+(a[9]*xl[1])+(a[8]*xl[2])+(a[6]*xl[4])+(a[0]*xl[1
0]) + (a[2]*xl[8]) +  (a[4]*xl[6]);
           result[1] =  (b[10]*yl[0]) + (b[8]*yl[2])+ (b[6]*yl[4]) +  (b[4]*yl[6]) +
(b[2]*yl[8]) + (a[7]*xl[3])+(a[5]*xl[5])+ (a[1]*xl[9]) + (a[3]*xl[7]);
                        result[1] = (-1) * result[1] ;

                        result1 = result1 + result[0] + result[1];
                                    if(result1< 0){ result1 = (-1) *result1; result1 =(
result1 /10000); yl[10] = result1; yl[10] = yl[10] * (-1); }
                                    else {result1 = result1/10000; yl[10] = result1;}
```

```
            if(inmemf(USER_REG, 5) != 0)
                  while(!aic23_write_data(yl[10]));   //if switch1 not pressed, play the
filtered tone

            for(i = 0;i<10;i++)
                        {xl[i] = xl[i+1]; yl[i] = yl[i+1];}
                         result1 = 0;


      }


}

// ---------------------------------------------------------------------------
// Main program
// ---------------------------------------------------------------------------

void main()
{
  int i , j;
  signed int tone,  nf;
  long signed temp1,temp;
  int temp3;
  signed int temp2;
  long unsigned phaseH, phase;
  unsigned long div = 1677216;
  int div2;
  long result;
  long a00,a11,a22,b11,b22;

  //filtering variables
  int r1,r2,r3,r4,r5,r;
  long long result1,result2,result3,result4,result5;
  // Initialize hardware
  hardware_init();

  //Start audio codec
  aic23_open();

  //initialization
  for(j =0 ;j<FREQ_RANGE;j++) {
        freq[j] = cycles[j];
        energy[j] = 0; }

  for(i = 0;i<11;i++) {
      xl[i] = 0;
      yl[i] = 0;
  }

  //aic23_write_cmd(AIC23_ANAPATH, 0x0010);
  //test_LED();
  //test_MIC();

  for(i = 0;i < FREQ_RANGE; i++){
      for(j = 0; j < cycles[i] ; j++){
            phaseH = 0;
            phase  = 0;
            outmemw(USER_REG,(freq[i] / 10));
            while(phase <= LUT_SIZE - 1){
```

```
                    phaseH = phaseH + delta_phase[i];
                    phase  = (phaseH >> 16);
                    if(phase > LUT_SIZE-1)break;
                    tone = lut[phase];
                    // Write left channel data
                    while (!aic23_write_data(tone));
                    while(!aic23_read_data(&temp2));
                    temp1 = (abs(temp2) * abs(temp2));
                    energy[i] =  energy[i] + temp1;
                    // Write right channel data
                    while (!aic23_write_data(tone));
                    while(!aic23_read_data(&temp2));

            }
        }
    }

    sort();
    for(i = 0;i<FREQ_RANGE; i++)
        temp1 = energy[i];
    outmemw(USER_REG,(freq[0]/10));
    for(i = 0; i<3 ;i ++){
        xl[i] = 0;
        yl[i] = 0;
        xr[i] = 0;
        yr[i] = 0;
    }
    while (inmemf(USER_REG, 4) != 0);          //wait for line in input
    aic23_write_cmd(AIC23_ANAPATH, 0x0010);
    //div1 = 32768;coefficients scaled by the factor 2^31, it is necessary to divide the
coefficients by div vallue first to avoid the out of range error
    nf = ((freq[0]/10) - 2);
    test_LineIn();
    aic23_close();

}
```