KEITHLEY

Application Note Series

Number 2297

Audio Analysis Testing Using a KPCI-3108 Board with LabWindows/CVI 5.0.1

Introduction

Manufacturers of cellular and cordless telephones employ a variety of methods to test the audio quality of their products during production. These tests may be as simple as measuring the Vrms of a sinewave input signal that is passed through the analog microphone and speaker circuits. In this case, a DMM usually offers sufficient measurement capability.

More detailed testing of the analog microphone and speaker circuit typically involves total harmonic distortion measurements, frequency response analysis, and noise measurements. For this scenario, a more specialized instrument is required for these audio measurements.

Some manufacturers make audio measurements through the complete phone circuit, including the digital encoder/decoder (CODEC) circuit and the RF circuitry. In this case, a communication analyzer is needed to receive and transmit the modulated RF signal from the phone under test (DUT). Speech CODECs are designed to process the multi-frequency content of human speech, so most of them do not recognize single frequency tones. Therefore, testing the complete path for audio signals in digital cellular phones usually requires a source capable of generating multiple frequencies or tones.

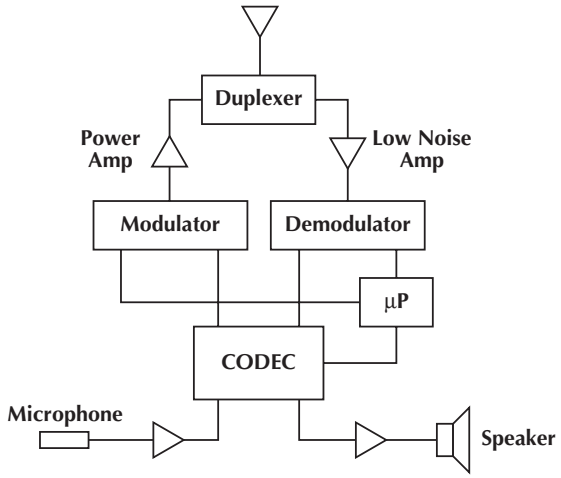

Figure 1. Block Diagram of a Cellular Phone

To qualify the audio circuit of a portable phone fully, an artificial "mouth" and an artificial "ear" are placed in front of the microphone and speaker of the wireless phone. The use of an artificial mouth and an artificial ear is a standard technique in design evaluation laboratories. Some manufacturers use these test system components in production. The mouth is connected to a multi-tone sinewave function generator. The ear receives the signal produced by the speaker. The output of the ear is acquired and is compared to the signal produced by the generator. The power spectrum and total harmonic distortion of this signal are computed. The frequency and amplitude of each harmonic of the output is compared to its counterpart in the generated wave. These tests are most often performed while the phone is in the RF transmission state.

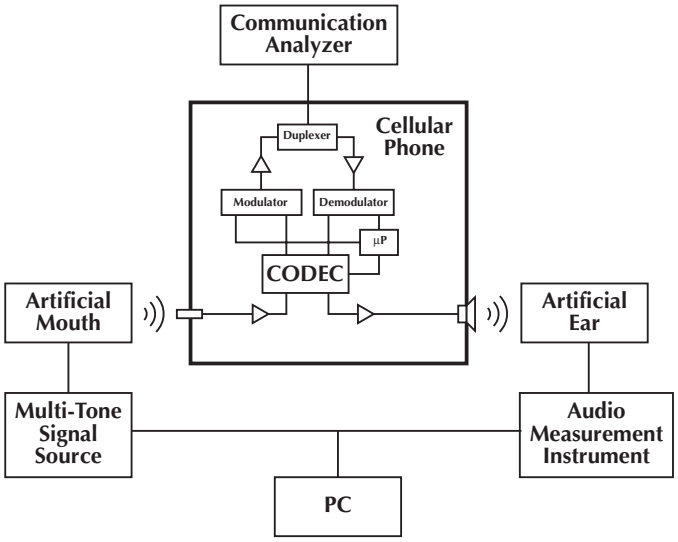

Figure 2. Audio Circuit Test System

Data Acquisition Approach

Basically, this application requires signal generation and measurement equipment, as well as software to control the hardware and perform the signal processing. A high performance data acquisition board offers a compact solution for sourcing and measurement. However, the data acquisition board must meet some important criteria:

- It must be able to acquire and generate signals simultaneously. Note that while these operations must be simultaneous, they do not necessarily need to be synchronized.
- It must have high A/D resolution and a high gain amplifier because the signal amplitudes are in the range of 100mV.
- The board must have an analog output capable of generating complex waveforms.

While Keithley offers a variety of data acquisition products that are suitable for this application, the KPCI-3108 High-Resolution Analog and Digital I/O Board is an excellent choice. It provides two waveform-quality analog outputs, allowing it to generate a two-tone waveform that simulates speech. The board's A/D converter samples signals at 100ksamples/sec with 16-bit resolution to provide accurate measurements in the audio band. In addition, the KPCI-3108 has 12 programmable gains, which yields resolution down to microvolts. Other good choices include the KPCI-3116 250kS/s Analog and Digital I/O Board or any of the ADwin Windows-based real-time measurement and control systems.

On the software side, data acquisition packages such as TestPoint™, LabVIEW, LabWindows/CVI, HP-VEE, etc., or applications created in programming languages such as Visual Basic, Visual C++, or Delphi provide the necessary power for audio analysis. However, it's important to note that, when a language is used, a mathematics library is necessary to compute the power spectrum. This library is available from companies such as Quinn-Curtis, Inc and AccuSoft Corporation. The data acquisition packages already include a mathematics library for performing the computations.

This application note outlines a solution for use in conjunction with the LabWindows CVI package, which is widely used in the telecommunications market.

While a DriverLINX® board driver can be used to control the KPCI-3108 board, a special LabWindows/CVI library has been designed to perform all the functions necessary to complete the tasks. The library can initialize the board, compute the waveform to be generated, acquire the signal, and compute the power spectrum and THD values.

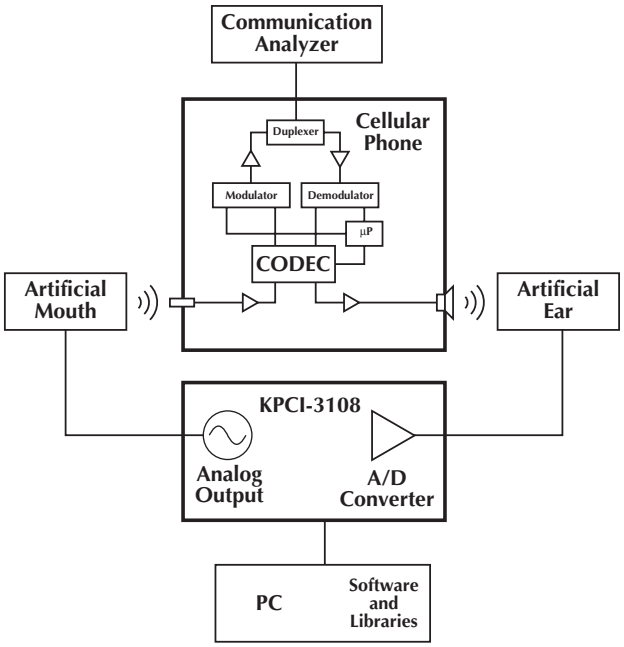

Figure 3. Audio Circuit Test System Using KPCI-3108

The Library

The KPCI-3108 LabWindows/CVI library for THD measurements supports all the functions necessary for the user to program the application without the need to work with the DriverLINX driver. The end user program must be linked with the regular DRVLINX32.LIB file that comes with the board. In order to use the library, the compiler must have the following constants defined or undefined:

/DWIN32_LEAN_AND_MEAN /D_MSC_VER=0

The library supports the following functions:

```
/*****************************************************************************/
/* Function : Init                                                           */
/* Purpose : This function must be called before any other functions in       */
/* the library. It is used to load the driver, init the driver, test the     */
/* board and initialize the hardware                                         */
/* Parameters :                                                              */
/*           nInitMethod : This parameter can take any of the following      */
/* values :                                                                  */
/*           KPCI3108_INIT_WITH_CVI_WIN_HWND:                                */
/*           The DLINX messages are sent to the main CVI window              */
/*           KPCI3108_INIT_WITH_CVI_THR_HWND:                                */
/*           The DLINX messages are sent to the main CVI window for the      */
/*           current thread.                                                 */
/*           KPCI3108_INIT_WITH_HIDDEN_PANEL:                                */
/*           The Dlinx messages are sent to a hidden private window that     */
/*           is created and deleted internally by this library               */
/* Return value :                                                            */
/*           Any of the possible library error codes. The function has       */
/*           been successful only when the returned value is                 */
/*           KPCI3108_NO_ERROR                                               */
/*****************************************************************************/
```

```
int KPCI3108_Init (int nInitMethod);
/*************************************************************************/
/* Function : Close                                                      */
/* Purpose : This function must be called during program termination to   */
/* close the driver.                                                     */
/* Parameters :                                                          */
/*            None                                                       */
/* Return value :                                                        */
/*           Any of the possible library error codes. The function has   */
/*           been successful only when the returned value is             */
/*           KPCI3108_NO_ERROR                                           */
/*************************************************************************/
int KPCI3108_Close (void);

/*************************************************************************/
/* Function : Start_ADTask                                               */
/* Purpose : Starts an acquisition on an analog input channel            */
/* Parameters :                                                          */
/*           ADChannel : Any AD channel from 0 to 7 in differential mode */
/* or from 0 to 15 in single ended mode                                  */
/*           nSamples : Number of sample to acquire (No limits except    */
/*           available memory).                                          */
/*           SamplingRate : Acquisition frequency in Hertz. From 1.0 to  */
/*           100000.0                                                    */
/*           ADRange : Range to be used, can take any of the following   */
/*           constants :                                                 */
/*               KPCI3108_RANGE_10V    : +/- 10 V.                       */
/*               KPCI3108_RANGE_5V     : +/- 5 V.                        */
/*               KPCI3108_RANGE_2_5V   : +/- 2.5 V.                      */
/*               KPCI3108_RANGE_1_25V  : +/- 1.25 V.                     */
/*               KPCI3108_RANGE_1V     : +/- 1 V.                        */
/*               KPCI3108_RANGE_500MV  : +/- 500 mV.                     */
/*               KPCI3108_RANGE_250MV  : +/- 250 mV.                     */
/*               KPCI3108_RANGE_125MV  : +/- 125 mV.                     */
/*               KPCI3108_RANGE_100MV  : +/- 100 mV.                     */
/*               KPCI3108_RANGE_50MV   : +/- 50 mV.                      */
/*               KPCI3108_RANGE_25MV   : +/- 25 mV.                      */
/*               KPCI3108_RANGE_12_5MV : +/- 12.5 mV.                    */
/* Return value :                                                        */
/*           Any of the possible library error codes. The function has   */
/*           been successful only when the returned value is             */
/*           KPCI3108_NO_ERROR                                           */
/*************************************************************************/
int KPCI3108_Start_ADTask (unsigned int ADChannel,
                           unsigned long nSamples,
                           double SamplingRate,
                           short ADRange);

/*************************************************************************/
/* Function : Acquire_AD                                                 */
/* Purpose : Starts an acquisition on an analog input channel            */
/* NOTE THAT THIS FUNCTION RETURN ONLY AFTER FULL COMPLETION OF THE      */
/* ACQUISITION TASK. DO NOT USE IT FOR LENGTHY ACQUISITIONS (MORE THAN   */
/* 10 Secs)                                                              */
/* Parameters :                                                          */
/*           ADChannel : Any AD channel from 0 to 7 in differential mode */
/* or from 0 to 15 in singled ended mode                                 */
/*           nSamples : Number of sample to acquire (No limits except    */
/*           available memory).                                          */
/*           SamplingRate : Acquisition frequency in Hertz. From 1.0 to  */
/*           100000.0                                                    */
/*           ADRange : Range to be used, can take any of the following   */
/*           constants :                                                 */
/*               KPCI3108_RANGE_10V    : +/- 10 V.                       */
/*               KPCI3108_RANGE_5V     : +/- 5 V.                        */
/*               KPCI3108_RANGE_2_5V   : +/- 2.5 V.                      */
/*               KPCI3108_RANGE_1_25V  : +/- 1.25 V.                     */
/*               KPCI3108_RANGE_1V     : +/- 1 V.                        */
/*               KPCI3108_RANGE_500MV  : +/- 500 mV.                     */
/*               KPCI3108_RANGE_250MV  : +/- 250 mV.                     */
/*               KPCI3108_RANGE_125MV  : +/- 125 mV.                     */
/*               KPCI3108_RANGE_100MV  : +/- 100 mV.                     */
/*               KPCI3108_RANGE_50MV   : +/- 50 mV.                      */
/*               KPCI3108_RANGE_25MV   : +/- 25 mV.                      */
/*               KPCI3108_RANGE_12_5MV : +/- 12.5 mV.                    */
```

```
/* Return value :                                                          */
/*           Any of the possible library error codes. The function has    */
/*           been successful only when the returned value is              */
/*           KPCI3108_NO_ERROR                                            */
/*************************************************************************/
int KPCI3108_Acquire_AD (unsigned int ADChannel,
                         unsigned long nSamples,
                         double SamplingRate,
                         short ADRange,
                         double* pData);

/*************************************************************************/
/* Function : Get_ADTask_Status                                          */
/* Purpose : This function return the actual status of an AD task        */
/* Parameters :                                                          */
/*           pStatus : On exit hold the actual task status               */
/*               KPCI3108_TASK_SCHEDULED : Task is about to start        */
/*               KPCI3108_TASK_ARMED : Task is ready and wait for trigger */
/*               KPCI3108_TASK_ACTIVE : Task is running                  */
/*               KPCI3108_TASK_INACTIVE : Task is stopped                */
/*               KPCI3108_TASK_DONE : Task completed                     */
/*           pCurrentBuffer : Actual task buffer under processing        */
/*           pNextElement : Next buffer element to be processed          */
/* Return value :                                                        */
/*           Any of the possible library error codes. The function has    */
/*           been successful only when the returned value is              */
/*           KPCI3108_NO_ERROR                                            */
/*************************************************************************/
int KPCI3108_Get_ADTask_Status(WORD* pStatus,
                               WORD* pCurrentBuffer,
                               DWORD* pNextElement);

/*************************************************************************/
/* Function : Get_ADTask_Data                                            */
/* Purpose : This function return the values sampled by an AD task       */
/* Parameters :                                                          */
/*           pData : On exit hold the actual task data                   */
/* Return value :                                                        */
/*           Any of the possible library error codes. The function has    */
/*           been successful only when the returned value is              */
/*           KPCI3108_NO_ERROR                                            */
/*************************************************************************/
int KPCI3108_Get_ADTask_Data(double* pData)

/*************************************************************************/
/* Function : Start_DATask                                               */
/* Purpose : This function start the generation of a waveform on any DA  */
/*            Channel. The wavefrom is built on 2 buffer that are sent one*/
/*            after each other until the Stop_DATask function is called  */
/* Parameters :                                                          */
/*           DAChannel : DA Channel to use : 0 or 1                      */
/*           pWaveform1 : Data buffer 1.                                 */
/*           pWaveform2 : Data buffer 2.                                 */
/*           nWaveformLenght : Size of the buffers                       */
/*           RefreshRate : Generation frequency in Hertz from 1 to 100000 */
/*           DARange : Range to be used, can take any of the following   */
/*           constants :                                                 */
/*                 KPCI3108_RANGE_10V    : +/- 10 V.                     */
/*                 KPCI3108_RANGE_5    : +/- 5 V.                        */
/* Return value :                                                        */
/*           Any of the possible library error codes. The function has    */
/*           been successful only when the returned value is              */
/*           KPCI3108_NO_ERROR                                            */
/*************************************************************************/
int KPCI3108_Start_DATask (unsigned int DAChannel,
                           double* pWaveform1,
                           double* pWaveform2,
                           unsigned long nWaveformLenght,
```

```
                            double RefreshRate,
                            short DARange);

/*****************************************************************************/
/* Function : Stop_DATask                                                    */
/* Purpose : This function stop the generation of a waveform on any DA       */
/* Channel                                                                   */
/* Parameters :                                                              */
/*           None                                                            */
/* Return value :                                                            */
/*           Any of the possible library error codes. The function has       */
/*           been successful only when the returned value is                 */
/*           KPCI3108_NO_ERROR                                               */
/*****************************************************************************/
int KPCI3108_Stop_DATask(void);

/*****************************************************************************/
/* Function : Get_ADTask_Status                                              */
/* Purpose : This function return the actual status of a DA task             */
/* Parameters :                                                              */
/*           pStatus : On exit hold the actual task status                   */
/*               KPCI3108_TASK_SCHEDULED : Task is about to start            */
/*               KPCI3108_TASK_ARMED : Task is ready and waait for trigger*/
/*               KPCI3108_TASK_ACTIVE : Task is running                      */
/*               KPCI3108_TASK_INACTIVE : Task is stopped                    */
/*               KPCI3108_TASK_DONE : Task completed                         */
/*           pCurrentBuffer : Actual task buffer under processing            */
/*           pNextElement : Next buffer element to be processed              */
/* Return value :                                                            */
/*           Any of the possible library error codes. The function has       */
/*           been successful only when the returned value is                 */
/*           KPCI3108_NO_ERROR                                               */
/*****************************************************************************/
int KPCI3108_Get_DATask_Status(WORD* pStatus,
                               WORD* pCurrentBuffer,
                               DWORD* pNextElement);

/*****************************************************************************/
/* Function : OnDLEvent                                                      */
/* Purpose : This callback function is called by the driver every time a     */
/* task event is encountered.                                                */
/* NOTE : This callback is declared but not defined. It is the job of the*/
/* library used to define its body.                                          */
/* Parameters :                                                              */
/*           Not relevant to the user, but must be passed without            */
/*           modifications to the PreProcessDLEvent function so the          */
/*           following global variables are                                  */
/*           updated according to the event :                                */
/*                  KPCI3108_wTaskID;                                        */
/*                  KPCI3108_wMessage;                                       */
/*                  KPCI3108_wDevice;                                        */
/*                  KPCI3108_wMode;                                          */
/*                  KPCI3108_wSubSystem;                                     */
/*                  KPCI3108_wBufferIndex;                                   */
/* Return value :                                                            */
/*           None                                                            */
/*****************************************************************************/
void CVICALLBACK KPCI3108_OnDLEvent(WinMsgWParam wParam,
                                    WinMsgLParam lParam,
                                    void *callbackData);

/*****************************************************************************/
/* Function : PreProcessDLEvent                                              */
/* Purpose : This function should be called to update the variables.         */
/*                  KPCI3108_wTaskID;                                        */
/*                  KPCI3108_wMessage;                                       */
/*                  KPCI3108_wDevice;                                        */
/*                  KPCI3108_wMode;                                          */
/*                  KPCI3108_wSubSystem;                                     */
/*                  KPCI3108_wBufferIndex;                                   */
/*           before processing a DL event                                    */
/* Return value :                                                            */
/*           None                                                            */
/*****************************************************************************/
```

```
void KPCI3108_PreProcessDLEvent(WinMsgWParam wParam,
                                WinMsgLParam lParam);

/*************************************************************************/
/* Function : UpdateDABuffer                                             */
/* Purpose : This function should be called to update any of the 2 DA    */
/* buffers while the DA task is running.                                 */
/* Parameters :                                                          */
/*           wBufNumber : 0 or 1                                         */
/*           pWaveform : Data buffer                                     */
/* Return value :                                                        */
/*           Any of the possible library error codes. The function has   */
/*           been successful only when the returned value is             */
/*           KPCI3108_NO_ERROR                                           */
/*************************************************************************/
int KPCI3108_UpdateDABuffer(WORD wBufNumber,
                            double *pWaveform);

/*************************************************************************/
/* Function : Set_DAOutput                                               */
/* Purpose : This function set the output of a DA channel to a specific  */
/*    level using a range.                                               */
/* Parameters :                                                          */
/*           DAChannel : DA Channel to use : 0 or 1                      */
/*           Value : Value to set to the DA channel in Volts from -10 to */
/*           10 Volts                                                    */
/*           DARange : Range to be used, can take any of the following   */
/*           constants :                                                 */
/*                 KPCI3108_RANGE_10V    : +/- 10 V.                     */
/*                 KPCI3108_RANGE_5    : +/- 5 V.                        */
/* Return value :                                                        */
/*           Any of the possible library error codes. The function has   */
/*           been successful only when the returned value is             */
/*           KPCI3108_NO_ERROR                                           */
/*************************************************************************/
int KPCI3108_Set_DAOutput(unsigned int DAChannel,
                          double Value,
                          short DARange);

/*************************************************************************/
/* Function : ResetWaveForm                                              */
/* Purpose : This function set the full content of the array pWaveform to*/
/* Parameters :                                                          */
/*           pWaveform : Array to reset                                  */
/*           nWaveformSize : Size of the array                           */
/* Return value :                                                        */
/*           None                                                        */
/*************************************************************************/
void KPCI3108_ResetWaveForm(double *pWaveform,
                            unsigned long nWaveformSize);

/*************************************************************************/
/* Function : AddHarmonicToWaveForm                                      */
/* Purpose : This function add a new harmonic to the array pWaveform     */
/* Parameters :                                                          */
/*           pWaveform : Array to reset                                  */
/*           nWaveformSize : Size of the array                           */
/*           RMS : RMS value of the newly added harmonic                 */
/*           Frequency : Frequency of the newly added harmonic           */
/*           GenFrequency : Frequency that will be used to generate the  */
/*           waveform on a DA output of the KPCI3108 board               */
/* Return value :                                                        */
/*           None                                                        */
/*************************************************************************/
void KPCI3108_AddHarmonicToWaveForm(double *pWaveform,
                                    unsigned long nWaveformSize,
                                    double RMS,
                                    double Frequency,
                                    double GenFrequency);
```

How to Use the Library

The library offers many different techniques to perform the required tasks, so individual users will find a variety of ways to perform the same task. The following approach is just one example of the possibilities.

Generating a multi-tone waveform from one of the analog outputs:

Before any measurements can be made, the board must generate a multi-tone sinewave. The library includes a full set of functions tailored to create the waveform, then output it in continuous mode.

First, we must choose some important values, particularly the generation refresh frequency, which is the duration of one full waveform. In this example, for the sake of simplicity, let's define the duration as one second. Therefore, the frequency (expressed in Hz) and the size of the array holding the waveform must be the same number. If we assume a duration of one second and a refresh frequency of 50kHz, then the waveform size would be 50,000 points.

Step 1: Declare, define, and initialize the array that will hold the waveform:

```
Double *pWaveform;
pWaveform = malloc( 50000 * sizeof(double) );
KPCI3108_ResetWaveform( pData, 50000 );
```

Step 2: Add all the harmonics to the waveform array:

To add harmonics to the waveform, a specific function named "AddHarmonicToWaveform" is used.

For this example, we will add three harmonics:

Harmonic 1: 50Hz @ 100mVRMS

Harmonic 2: 217Hz @ 37mVRMS

Harmonic 3: 1kHz @ 25mVRMS

This results in the following piece of code:

```
KPCI3108_AddHarmonicToWaveForm(pWaveform, 50000, 0.1,
50.0, 50000.0) ;
KPCI3108_AddHarmonicToWaveForm(pWaveform, 50000,
0.037, 217.0, 50000.0) ;
KPCI3108_AddHarmonicToWaveForm(pWaveform, 50000,
0.025, 1000.0, 50000.0) ;
```

Step 3: Output the signal:

This final step is easy. Just use the "Start_DA" function, and it's done. Use the "Stop_DA" when you want to stop generating the signal, either at the end of the test or if a change in the waveform is required.

```
KPCI3108_Start_DATask (0, pWaveform, pWaveform, 50000,
50000.0, KPCI3108_RANGE_5V) ;
…/…
KPCI3108_Stop_DATask ();
```

Acquiring the signal:

Now that the signal is present on the analog output, we can acquire the signal returned by the phone through the artificial ear. To acquire the whole waveform, we will sample the points during one second. In order to use a FFT algorithm to compute the power spectrum and ultimately the THD, the size of the acquisition should be a multiple of 2^n. (The computations of FFT algorithms are optimized for and based on a multiple of 2^n samples.) As explained previously, this implies that the number of samples is equal to the acquisition rate. In this example, we will perform an acquisition of 16,384 samples at a rate of 16,384Hz.

To do this job, we simply have to define and declare an array that will hold the data, then call the "Acquire_AD" function as shown in this example:

```
double* pData;
pData = (double *)malloc((16384)*sizeof(double));
ErrCode = KPCI3108_Acquire_AD(0, 16384, 16384.0,
KPCI3108_RANGE_250MV, pData);
```

Computing the power spectrum and THD:

After completion of the acquisition, all that remains is to compute the power spectrum and the THD. The following is a documented example of how to perform these tasks while taking advantage of the LabWindows CVI math function (AutoPowerSpectrum):

```
// Declare arrays to hold generated frequencies and
amplitudes
// These variables should be filled before each calls to
AddHarmonic
// In this example, the pFreqs array hold 3 values :
50,217,1000
// The pRMS array hold 3 values : 0.1, 0.037, 0.025
// nFreq take the value 3
double pFreqs[1024];
double pRMS[1024];
WORD nFreq;

// Define variables used for computation
double* pSpectrum;
double df;
unsigned long i_loop;
unsigned long j_loop;

// Define the pSpectrum array
pSpectrum = (double *)malloc((16384/2)*sizeof(double));

// Compute the power spectrum
AutoPowerSpectrum (pData, 16384, 1.0/16384.0 ,pSpectrum ,
&df);
for(i_loop = 0; i_loop < (16384/2); i_loop++)
        {
        *(pSpectrum + i_loop) =  sqrt(*(pSpectrum + i_loop));
        }
```

```
// Get fundamental and tone harmonics amplitudes
FundamentalAmp = 0.0;
for(i_loop=0;i_loop<nFreq;i_loop++)
        {
        HarmonicMax[i_loop]=0.0;
        for(j_loop=(unsigned long)floor(pFreqs[i_loop])-
1;j_loop<(unsigned long)ceil(pFreqs[i_loop])+1;j_loop++)
        {
        HarmonicMax[i_loop] = (HarmonicMax[i_loop] >
*(pSpectrum + j_loop)) ?  HarmonicMax[i_loop] : *(pSpectrum +
j_loop);
        }
        FundamentalAmp = (FundamentalAmp >
HarmonicMax[i_loop]) ? FundamentalAmp : HarmonicMax[i_loop];
        }

// Compute THD
SumOfHarms=0.0;
for(i_loop=0;i_loop<(ACQ_SIZE/2);i_loop++)
        {
        if(*(pSpectrum + i_loop) != FundamentalAmp)
        SumOfHarms+=((*(pSpectrum + i_loop)) * (*(pSpectrum +
i_loop)));
        }
THD = sqrt(SumOfHarms) / FundamentalAmp;
```

For more information or assistance in implementing this application, contact Alain Lainé, Keithley Instruments France. Phone: 33-1-60-11-51-55. Fax: laine_alain@keithley.com

KEITHLEY

Keithley Instruments, Inc. 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168
1-888-KEITHLEY (534-8453) www.keithley.com

BELGIUM:	**Keithley Instruments B.V.**	Bergensesteenweg 709 • B-1600 Sint-Pieters-Leeuw • 02-363 00 40 • Fax: 02/363 00 64
CHINA:	**Keithley Instruments China**	Yuan Chen Xin Building, Room 705 • 12 Yumin Road, Dewai, Madian • Beijing 100029 • 8610-6202-2886 • Fax: 8610-6202-2892
FRANCE:	**Keithley Instruments Sarl**	3, allée des Garays • 91127 Palaiseau Cédex • 01-64 53 20 20 • Fax: 01-60 11 77 26
GERMANY:	**Keithley Instruments GmbH**	Landsberger Strasse 65 • 82110 Germering • 089/84 93 07-40 • Fax: 089/84 93 07-34
GREAT BRITAIN:	**Keithley Instruments Ltd.**	The Minster • 58 Portman Road • Reading, Berkshire RG30 1EA • 0118-9 57 56 66 • Fax: 0118-9 59 64 69
INDIA:	**Keithley Instruments GmbH**	Flat 2B, WILLOCRISSA • 14, Rest House Crescent • Bangalore 560 001 • 91-80-509-1320/21 • Fax: 91-80-509-1322
ITALY:	**Keithley Instruments s.r.l.**	Viale San Gimignano, 38 • 20146 Milano • 02-48 39 16 01 • Fax: 02-48 30 22 74
KOREA:	**Keithley Instruments Korea**	2FL., URI Building • 2-14 Yangjae-Dong • Seocho-Gu, Seoul 137-130 • 82-2-574-7778 • Fax: 82-2-574-7838
NETHERLANDS:	**Keithley Instruments B.V.**	Postbus 559 • 4200 AN Gorinchem • 0183-635333 • Fax: 0183-630821
SWITZERLAND:	**Keithley Instruments SA**	Kriesbachstrasse 4 • 8600 Dübendorf • 01-821 94 44 • Fax: 01-820 30 81
TAIWAN:	**Keithley Instruments Taiwan**	1FL., 85 Po Ai Street • Hsinchu, Taiwan, R.O.C. • 886-3-572-9077• Fax: 886-3-572-9031

No. 2297
10110KAP