The PC Sound Card as a Voltage Measurement Platform in Power System Applications

by

Joel Allen Killingsworth

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
December 13, 2014

Keywords: Sound Card Voltage Measurement,
Voltage Harmonic Measurement, IEC 61000-4-7

Abstract

In this thesis document, the feasibility of using a PC sound card to replace traditional data acquisition (DAQ) devices for the measurement of voltage signals in power system applications is discussed. Communication between software applications and sound card hardware is established through the use of the PortAudio software library for audio applications. A PC sound card is tested using the example application of voltage harmonic measurement structured to be consistent with the requirements found in IEC standard 61000-4-7. It is found that a PC sound card is capable of continuous recording under the load of heavy calculation and signal processing for an indefinite period of time.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ADC	Analog to Digital Converter
API	Application Programming Interface
DAQ	Data Acquisition
DFT	Discrete Fourier Transform
DMM	Digital Multi-Meter
DTFT	Discrete-Time Foutier Transform
FFT	Fast Fourier Transform
FGEN	Function Generator
WASAPI	Windows Audio Session API
WMME	Windows Multi-Media Extensions

Chapter 1: Introduction

Electric power is central to modern culture and industry. Ensuring that electrical power is provided in a reliable fashion is, therefore, of the utmost importance. In order to ensure proper operation of a power system, it is necessary to monitor the condition of that system. If the system is pushed by negligence beyond what the components can handle, costly damage and failure of electric service can result.

The condition, or state, of a power system refers to the voltage at every node in the system. These are the basic physical quantities that are present. They are complex values consisting of magnitude and phase. If every complex node voltage is known and the impedance of every system element is known, then all other quantities of interest can be found using basic circuit analysis techniques. Current, real power, reactive power; knowledge of all of these quantities is vital in maintaining a reliable power system, and all of it starts with voltage.

When estimating the state of a power system, many node voltages must be guessed. The fewer guesses that have to be made, the more accurate the final understanding of the state of the power system. Measuring voltage is, therefore, an essential part of running and maintaining a reliable electric power system.

Many companies exist which manufacture and sell equipment for the purpose of measuring voltage. These voltage measurement devices come in all shapes and sizes and are rated for many different ranges of voltage. These devices will be referred to generally as data acquisition (DAQ) devices. Modern DAQ devices commonly use analog to digital converters (ADC) to convert analog values such as voltage into digital values. Once converted, these digital values can be used by computer systems to perform whatever calculations are necessary for the application.

It is possible, however, to remove the DAQ device from this process. Because computers often must be used in conjunction with DAQ devices, it is financially advantageous to implement the functionality of a DAQ device using hardware that already exists on a computer. The hardware component that can be most readily applied to this problem is the sound card.

Sound cards are peripheral devices for computer systems that process audio information. Audio information is sent to speakers or headphones to produce sound. Also, audio information can be recorded to a file when acquired from a mic. This recording capability utilizes an ADC to measure AC voltage signals in exactly the same manner as a DAQ device. Therefore, software can be developed which replaces the traditional DAQ device with a sound card.

Sound cards come standard on the vast majority of personal computers and are, therefore, more economically advantageous than a DAQ device, provided that certain minimum performance requirements are met. Many of these requirements are specific to the application and cannot be addressed in a general discussion. However, there are some general requirements that can be addressed. The sound card must be capable of continuous recording while the controlling software performs whatever calculations are required in real time. In addition, this continuous operation must be sustainable for an indefinite period of time.

In order to determine if these requirements are achievable using a PC sound card, it is necessary to perform tests on specific software and hardware components. Communication with the hardware must be established, followed by determining the electrical characteristics of the recording device on the sound card. Finally, processing and recording experimental data is necessary to determine the fitness of the sound card for use as a voltage measurement platform.

Chapter 2: Software Characterization

In order to use a PC sound card for voltage measurements, it is first necessary to understand how to communicate with the sound card hardware. In a Windows operating system environment, peripheral hardware components, such as sound cards, are controlled through software packages known as drivers. Applications developed by programmers can interface with these drivers in order to communicate with the hardware they control.

2.1: Drivers

Drivers are made up of two layers. There is the layer which communicates directly with the sound card. This layer is known as the driver kernel. The second layer communicates with a program. This layer is known as the driver application programming interface (API) [1]. This hierarchy is shown in Figure 1.

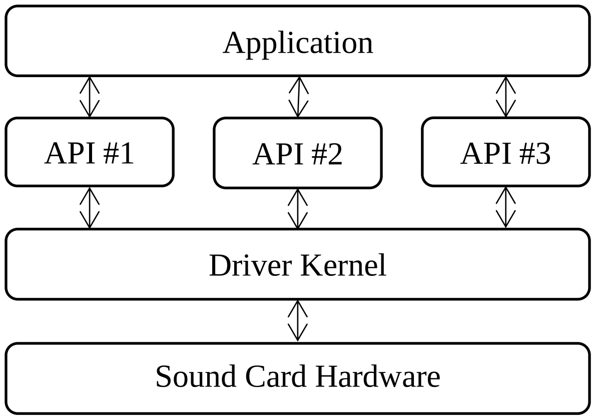

Figure 1: Anatomy of a Driver

The driver API provides a list of commands that can be accessed by programs. Those commands are then translated by the API and passed to the driver kernel to execute on the sound card hardware. Because of this two layer design, a single driver kernel can be accessed by more than one API, allowing a variety of interface configurations at the program level.

2.2: PortAudio

Developing the interface between an application and the driver API that is to be utilized is a non-trivial task. Fortunately, third party libraries already exist for this purpose. One such library is called PortAudio.

PortAudio provides a third level for interfacing with sound cards, as is shown in Figure 2 [2]. This extra level allows the programmer to use a set of C++ commands to access the desired API rather than having to conform to the specific API commands themselves. Moreover, as is indicated in the diagram, PortAudio is capable of interfacing with more than one API. This capability allows the programmer to develop an application without knowing exactly what API is going to be eventually used. This selection can easily be made at run-time and need not be pre-determined.

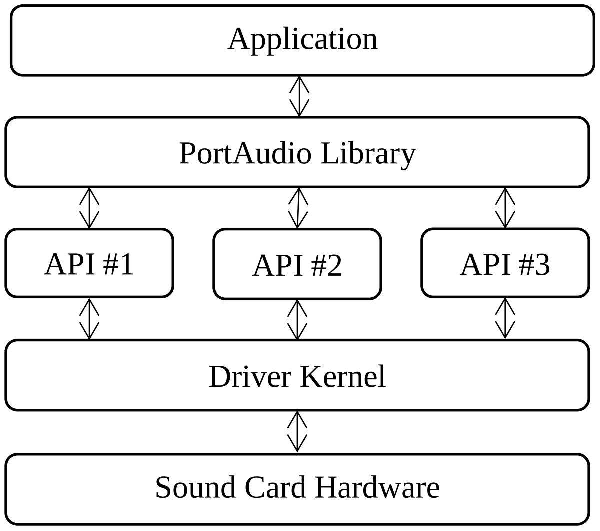

Figure 2: PortAudio and Drivers

Windows APIs that are natively supported by PortAudio include Windows Multi-Media Extensions (WMME), DirectSound, and Windows Audio Session API (WASAPI) [2]. WMME has been standard on all Windows machines since Windows 3.1 [1]. DirectSound is a part of the DirectX package which is most often used as the driver platform of choice for game developers since Windows 95 [1]. The DirectSound API is, therefore, available to all modern Windows computers. However, there is no guarantee that the DirectX package will have been installed on a given machine. WASAPI was created as a replacement for WMME and has been standard on all Windows machines since Windows Vista [3].

PortAudio accesses the audio card hardware using a software construct known as a stream. Streams are specialized objects in an application's code which determine the size, shape, and type of data that is to be passed to, or taken from, the hardware. There are many parameters that go into defining a stream. A few of the more important parameters are the number of input/output channels, the format for the data being streamed, and the length of the stream's buffer [2].

A PortAudio stream can be set up to take input and provide output simultaneously. Alternatively, a stream can be purely input or purely output. The number of data channels that can be opened in either direction is dependent on hardware.

The data format parameter for PortAudio streams can be 32-bit floating point, 32-bit, 24-bit, 16-bit, and 8-bit integer, as well as 8-bit unsigned integer. The values available for each of the integer data formats are merely the range of values inherent to that data type. In the case of the 8-bit unsigned integer format, a value of 128 is considered the reference value, or ground. The range of values available to the 32-bit floating point format is +1.0 to -1.0 [2].

In a PortAudio stream, the buffer length parameter determines how many data points are passed to or from the hardware at a time. This value can be automatically assigned by the PortAudio library to optimize performance. However, if the buffer length is automatically assigned, the length may vary over time. In the event that an application requires a specific buffer length, the length can be specified to be any arbitrary integer value [2].

There are two methods of communication between a stream and an application. The first method is by way of a callback function. This method raises an interrupt every time the streams buffer fills, for input, or empties, for output. A developer defined function is then run to collect and pass on whatever information is necessary. This function operates at a high priority and must resolve as quickly as possible in order to avoid interfering with other operations. Therefore, the callback function is prohibited from containing any memory allocation or file writing instructions. Failure to abide by this rule can lead to runtime errors [2].

The second stream communication method provided by the PortAudio library is a blocking scheme. This method entrusts the timing of communication to the application rather than to interrupts. This means that an input stream buffer will fill and then wait for a command from the application to be read. The same is true of an output stream [2]. It is important to note that this stream communication method cannot guarantee continuous operation.

When declaring a stream using the PortAudio library, an endpoint device must be selected. What would normally be considered a single piece of hardware may have several endpoint devices. A sound card, for example, may have a headphone jack and a mic jack, just to name two. Each of these endpoint devices is, for the purposes of stream communication, considered to be a completely separate device.

Furthermore, each of these endpoint devices may be compatible with commands from more than one API. In that event, the PortAudio library enumerates an instance of the endpoint device for each API that could be used to communicate with it. For example, if the mic jack on a sound card could be controlled using either WMME or WASAPI, the PortAudio library would recognize two different mic jacks. One of these jacks would be controlled using WMME and the other using WASAPI.

Once a PortAudio stream has been initialized, communication with an endpoint device can be established. It is then up to the programmer to dictate what sort of information is communicated. At this point in the process, the programmer can proceed with whatever processing the data requires.

Chapter 3: Voltage Measurement Platform

Once communication has been established with a sound card device, testing can begin. For use as a voltage measurement platform, it is necessary that an application should first be able to record data coming into the sound card. Beyond that, certain electrical and operational characteristics must be known about the device. These characteristics include such things as transfer characteristics, frequency response, and whether or not the hardware can take measurements continuously.

3.1: Hardware

The PC soundcard used for this experiment was integrated on the motherboard of a Dell Studio SPX laptop manufactured in 2009. As such, the electrical specifications for the sound card were not available for reference. This fact meant that all pertinent specifications had to be acquired through experimentation. The electrical characteristics that were determined experimentally include available sampling frequencies, input voltage to output value transfer curve, and magnitude frequency response.

The specific hardware component on the sound card that was tested was the mic input jack. This port accepts a standard one-eighth inch audio cable, or TRS cable. A TRS cable has three contacts: a ground or reference line, and two separate signal channels. The mic input works on measuring a voltage signal using an ADC. Using this hardware for measurement provides two input channels for voltage signals.

3.2: Experimentation Tools

In order to determine the electrical characteristics of the mic input jack, a signal generator was required. The signal generator used for this experiment was an NI ELVIS II+ prototyping board. This device houses, among other things, a digital function generator.

According to the user's manual for the ELVIS board [4], the ELVIS board's function generator (FGEN) is capable of supplying a sinusoidal signal with peak-to-peak amplitude of up to 10 V and a frequency of up to 50 MHz decimated by a 28-bit frequency divider for a frequency step of 0.18626 Hz. This theoretical step size was validated by observation. The frequency of the output signal varied by no more than 0.1 Hz at any point during experimentation.

3.3: PortAudio Choices

Once a signal generator was acquired, the next step was to set up a recording program. The mic input jack provides two input channels that the PortAudio library can access. For the purposes of this experiment, only one of these channels was used. The data format selected for recording by the PortAudio library was 32-bit floating point. As mentioned previously, this data format has a maximum value of +1.0 and a minimum value of -1.0.

The mic input jack is capable of receiving commands through more than one API. Two of the APIs that could access this endpoint device were WMME and WASAPI. Consequently, the PortAudio library enumerated two separate mic input jacks, one through WMME and the other through WASAPI.

3.4: Sampling Rate Capabilities

Both mic input jack endpoint devices were tested for the range of available sampling frequencies using the built in functions of the PortAudio library. The endpoint devices were selected by input streams and then tested with multiple sampling frequencies until the full range for each was found. The full range of available sampling frequencies for the WMME API was found to be 1 kHz to 200 kHz. On the other hand, WASAPI provided 44.1 kHz as the only available sampling frequency.

A device's list of available sampling frequencies is limited by the device's clock frequency. Therefore, the lists of available sampling frequencies for both WMME and WASAPI indicate that WMME automatically performs sample rate conversion and WASAPI does not. This supposition was supported by literature review [3]. Furthermore, this result also indicates that the oscillator on board the soundcard operates at an integer multiple of 44.1 kHz.

3.5: Transfer Curve

In order to find an input voltage to output 32-bit floating point number mapping curve, the ELVIS board's FGEN was used to generate a sinusoidal signal with a frequency of 60 Hz. This frequency was chosen because it is a frequency of primary interest in power system measurements. The amplitude of the input sinusoid was swept and the results are shown in Figure 3. The amplitude and frequency of the FGEN output was verified using an oscilloscope also housed on the ELVIS board. These results clearly indicate a linear relationship between input and output with maximum meaningful input voltage amplitude of approximately 135 mV. It is clear from Figure 3 that the tested sound card has a built-in gain of approximately 7.25 V^{-1}.

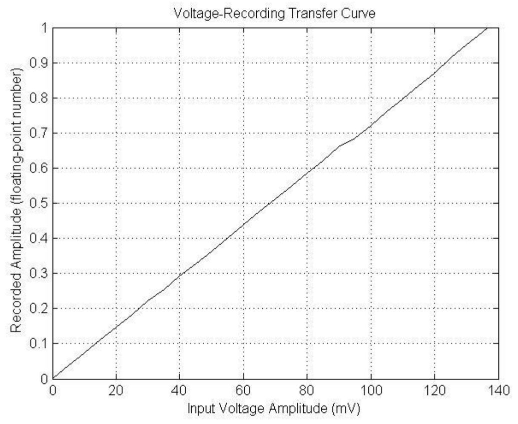

Figure 3: Input Voltage to Output Value

3.6: Magnitude Frequency Response

The next step in the characterization of the sound card mic input jack was to find the frequency response. Frequency response characterization is accomplished by passing a set of input signals through a system, recording the output, and comparing the two. This comparison is performed by way of the Fourier transform and results in both a magnitude versus frequency curve and a phase versus frequency curve. Only the magnitude portion of the frequency response was considered in this experiment. It is important to note that the phase response would also be useful information. However, acquiring the phase response data would require more precise timing information than was available for the input and output signals.

A 100 mV amplitude sinusoidal signal was used as the input for this test. The frequency of this signal was swept from 0 to 25 kHz, again verified by the ELVIS board's oscilloscope. The results of this sweep were compiled to form a magnitude response plot. The plotted values were normalized by the maximum recorded value. They were also converted to dB for ease of reading. The resulting plot is shown in Figure 4.

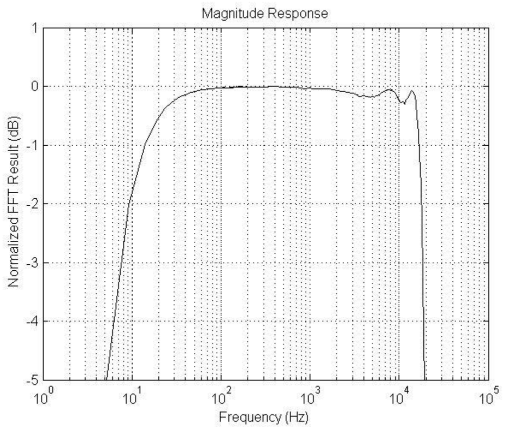

Figure 4: Magnitude Response

It is clear from the plot in Figure 4 that the mic input jack acts as a band-pass filter. The cutoff frequencies of filters are typically defined at the -3 dB points on the magnitude response plot. Using this definition, this band-pass filter appears to operate from roughly 7 Hz to nearly 20 kHz.

3.7: Continuity Testing

Once characterization of the hardware was complete, the next step was to determine if the hardware would be capable of continuous recording. Many recording applications require that the recording be continuous, without any jumps or skips. In order to test for continuity of data, a sinusoidal signal was input to the mic jack. In software, using the PortAudio library, the derivative of the collected data was calculated and recorded to a file for later review. This process was to be repeated for each buffer's worth of recorded data.

Because the derivative of a sinusoidal signal is another sinusoidal signal, the maximum value of the derivative is known. If any skips in the data were to occur, a discontinuity would be produced in the recorded signal which would cause the recorded derivative value to exceed its theoretical maximum. Therefore, any spikes in the filed derivative information would indicate a discontinuity in the recorded data.

Two continuity tests were conducted on the mic input jack. The first was using the WMME API. The input signal was a 60 Hz sinusoid with a peak-to-peak voltage of 0.2 V generated by the ELVIS board’s FGEN. The signal was sampled at 7.68 kHz so that there would be 128 samples per 60 Hz cycle. The buffer was set to 1280 points. This length corresponds to ten 60 Hz cycles, or approximately 0.167 s. The derivative was calculated from the recorded float values in real time and scaled so that the resulting sinusoid had the same amplitude as the recorded float value sinusoid.

The continuity test ran for 2 hours and 22 minutes before a buffer overflow error was encountered and the program stopped recording. Buffer overflow is merely when the stream attempts to write new data to the application’s buffer before the calculations on the old set of data have been completed. This is a common problem in real-time calculation systems and was caused, in this case, by the lengthy process of writing data to a file. The typical solution for this issue is to increase the length of the buffer so that there is more time in between buffers to record the data to a file. This solution also increases the time needed to perform calculations, but not nearly as much as it increases the available time.

Upon examination of the data file recorded for the WMME continuity test, it was clear that some data had indeed been skipped. As previously mentioned, WMME performs sample rate conversion automatically. The time delay between signal input and recorded data point, or latency, is, therefore, relatively high. This high latency can result in periodically dropping values in order to catch back up with the real-time input. The spikes in the derivative data are shown in Figure 5. These spikes were identified by finding data points with an absolute value greater than approximately 0.725 as dictated by the 0.2 V peak-to-peak input along with the gain of 7.25 V^{-1} depicted in Figure 3.

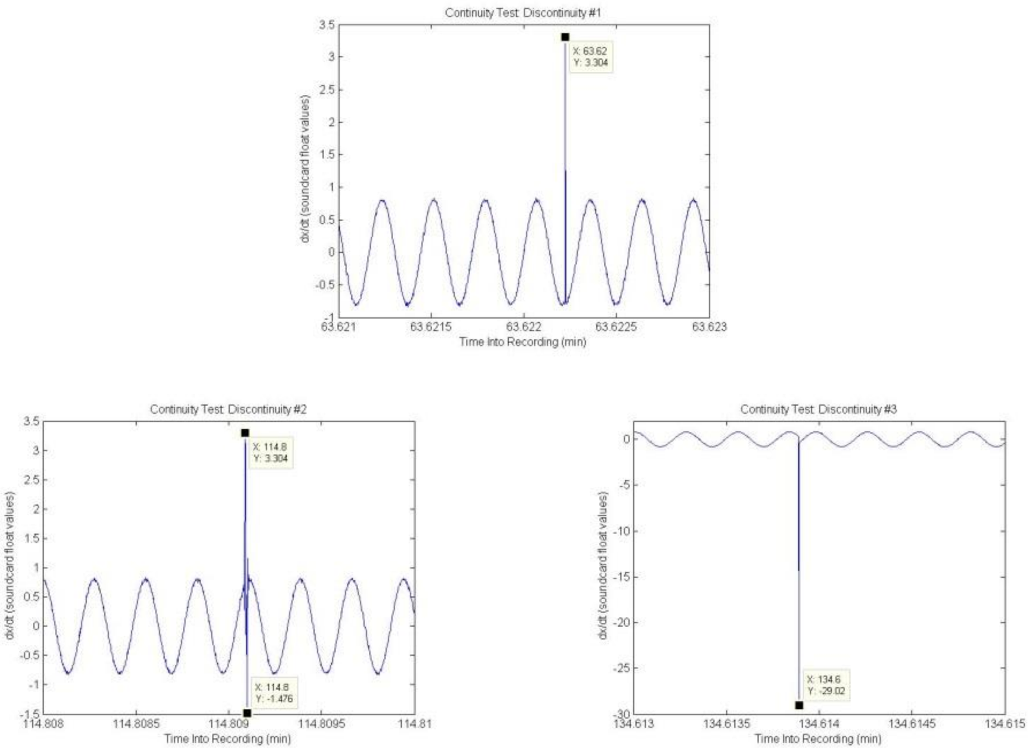

Figure 5: WMME Continuity Results

The second continuity test was run using WASAPI to access the mic input jack. The test signal was the same as before. The only difference was that the hardware was sampling at 44.1 kHz. Once the signal was sampled, linear interpolation was used to achieve a virtual sampling rate of 7.68 kHz, the same rate as the previous test. The derivative operation was then performed on the virtually sampled signal. Also, the buffer was extended to accommodate one second of data in order to eliminate the buffer overflow issue.

This second continuity test ran for 8 hours and 24 minutes. It was ended voluntarily without encountering any errors. Upon examination of the recorded derivative data file, no spikes were found, meaning that there were no skipped samples for the entire duration of the experiment.

This result clearly indicates the superiority of accessing hardware through WASAPI over WMME. Since WASAPI does not perform any sample rate conversion, the latency of the recording operation is relatively low and no samples are skipped. The assurance of continuity using WASAPI does come at a price, however. Every buffer's worth of data must be interpolated to the appropriate sampling frequency by the application. This extra action uses valuable processing time before the next buffer's worth of data is ready for processing.

Chapter 4: Signal Processing Principles

Once all of the necessary characteristics have been established, the sound card is ready for use as a voltage measurement platform. However, in order to demonstrate the capabilities of this measurement device, an example application has been selected. The application that will be explored is that of harmonic measurement of voltage in the power system.

As is implied by the name, harmonic measurements deal with signals containing multiple frequencies. In order to properly handle the measurement of these signals and any associated calculations, it is important to understand a few concepts in the field of digital signal processing.

When it comes to measurement and instrumentation, digital signal processing is a field of great importance. The ability to use computers to process data rather than relying on analog components provides a large degree of flexibility. Designs can be altered on the fly by simply changing a line of code, whereas changing out hardware components is a much more involved process.

4.1: Fourier Series and the Fourier Transform

The first and most important topic for discussion will be the Fourier transform. The Fourier transform is related to the Fourier series which is a concept whereby any periodic signal can be represented as an infinite sum of sinusoids. Each of these sinusoidal components has its own frequency which is an integer multiple of the frequency of repetition in the original periodic signal. The relative magnitude and phase angle of each of these sinusoids are what determine the overall shape of the original periodic signal. The Single-Sine form of the Fourier series is defined as is shown in (1) where the c_0 term represents the average value of the original signal, the c_k and φ_k terms are the amplitude and phase angle of each of the sinusoidal components, and the ω_1 term is the radian frequency of the original periodic signal. Each multiple of that frequency is called a harmonic [5]. Alternatively, the Fourier series can be expressed in a sine-cosine form as shown in (2).

$$f(t) = c_0 + \sum_{k=1}^{\infty} c_k \sin(k\omega_1 t + \varphi_k) \tag{1}$$

$$f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(k\omega_1 t) + b_k \sin(k\omega_1 t) \tag{2}$$

The series shown in (1) relates amplitudes and angles to specific frequencies. The Fourier transform can be used to find these quantities. This transform converts a continuous-time domain signal into a continuous-frequency domain signal. Example signals before and after applying the Fourier transform are shown in Figure 6. From this result, the harmonic information shown in (1) can be acquired.

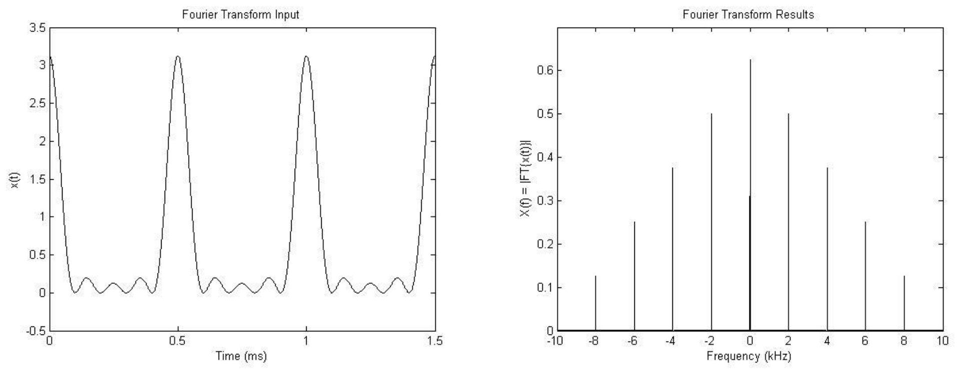

Figure 6: Fourier Transform Example

The Fourier transform holds for continuous-time signals. However, in the digital domain, all signals are discrete-time. For such signals, the discrete-time Fourier transform (DTFT) must be used. The DTFT takes a discrete-time domain signal and generates a continuous-frequency domain signal. As is shown in Figure 7, the results of the DTFT are periodic whereas the results of the Fourier transform are not. This fact leads to the potential for aliasing which will be discussed later.

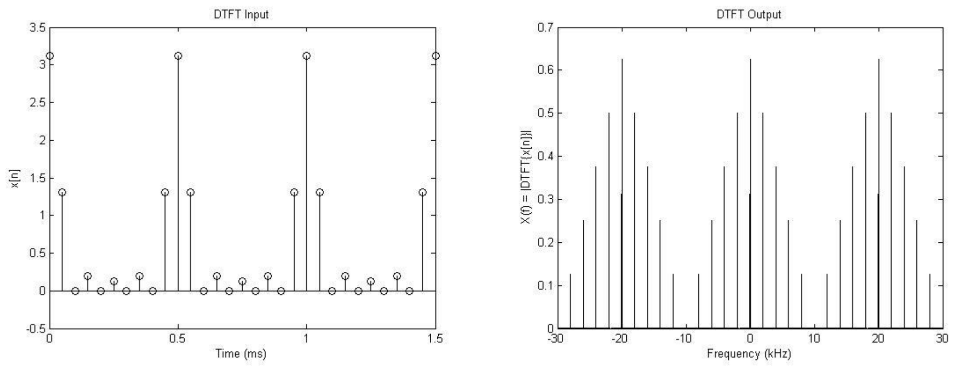

Figure 7: DTFT Example

The DTFT can operate on discrete-time signals. However, because the result is still a continuous signal, it cannot be computed by a digital device. To solve this issue, the discrete Fourier transform (DFT) is used. The DFT takes a discrete-time domain signal and generates a discrete-frequency domain signal. All signals involved are discrete in nature; therefore the results of this transform can be accurately calculated using a computer. As is shown in Figure 8, the periodicity of the continuous DTFT results is still present in the discrete DFT results. It is helpful to think of the DFT results as being a sampled version of the DTFT results.

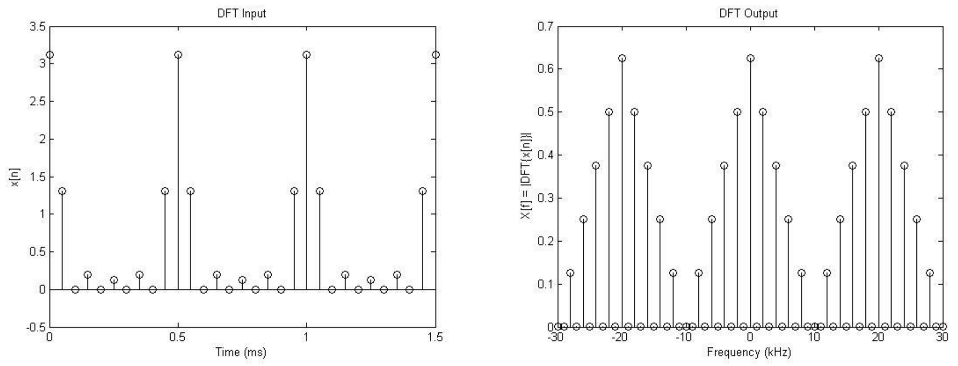

Figure 8: DFT Example

The DFT is completely calculable by a computer. However, completing the full calculation is computationally intensive. The periodic and symmetric nature of the result can be exploited to simplify the calculations involved. When this approach is taken, the resulting algorithm is called a fast Fourier transform (FFT).

It is important to note that the total number of points operated on using most FFT algorithms must be a power of two. Other than this limitation on the size of the input sequence, use of an FFT algorithm does not affect the frequency domain result. Mathematically, FFT algorithms are identical to the DFT; they are just less computationally intensive.

4.2: Discrete Fourier Transform Concerns

There are some practical concerns that must be addressed regarding the use of an FFT algorithm to analyze the frequency content of a signal. First, there is the concept of frequency resolution. Because the FFT produces a discrete-frequency domain signal, there must be some distance in the frequency spectrum between consecutive points in the calculation result. The closer these points are together in the frequency spectrum, the greater the frequency resolution.

Frequency resolution is determined by the sampling frequency of the discrete-time domain signal and the total number of points being used [6]. This relationship is shown in (3) where f_B is the spacing between points, f_s is the sampling frequency used to sample the discrete-time domain signal from the original continuous-time domain signal, and M is the total number of points being used.

$$f_B = \frac{f_s}{M} \tag{3}$$

Inspection of the relationship in (3) reveals that the spacing between points is the inverse of the length of time represented by the time domain samples being used, or window width (T_N). This relationship is demonstrated in (4) where T_s is the sampling period and is the inverse of f_s. Frequency resolution is, therefore, directly controlled by the length of time that is used as input for the FFT operation.

$$f_B = \frac{f_s}{M} = \frac{1}{MT_s} = \frac{1}{T_N} \tag{4}$$

This spacing between points in the frequency domain result can cause problems if some of the frequency content of the time domain signal does not fall exactly where a point is defined to represent it. For example, if a signal has energy at 8 Hz and an FFT is performed on the signal with 5 Hz between each resulting point, the result would have information available at 5 Hz and 10 Hz, but not at 8 Hz. The FFT would spread the 8 Hz energy to the surrounding available points. It is helpful to think of each point as a bin in which a range of possible frequencies collect. In this example, some of the 8 Hz energy would show up in the 5 Hz bin and the 10 Hz bin or even beyond. This phenomenon is known as frequency bleeding [6]. It is therefore necessary to make sure that all frequencies of interest in a given time domain signal will fall neatly into a frequency bin and will not be left to bleed out over several bins.

Another issue, brought about by the fact that the FFT operation is completely calculable, is that the results of the FFT do not extend to infinite frequency. There is a maximum frequency that can be represented by a given FFT operation. This maximum frequency is equal to $f_s \div 2$ [6]. Because the FFT results are periodic, as is shown in Figure 8, if the time domain signal were to contain energy at frequencies higher than $f_s \div 2$, the spectral copies would begin to overlap. This overlapping is called aliasing [7]. The minimum sampling rate to prevent aliasing is twice the highest frequency component of the signal to be sampled. This minimum rate is known as the Nyquist rate [7].

For example, the Fourier transform of a continuous-time domain signal ($x(t)$) produces a continuous-frequency domain signal ($FT\{x(t)\}$) with content up to the maximum frequency (f_{max}) of the original signal. A representation of $FT\{x(t)\}$ is shown in Figure 9. Now, assume that $x(t)$ is sampled at the Nyquist rate to generate the discrete-time signal $x_1[n]$. The DTFT of this new signal ($DTFT\{x_1[n]\}$) would not overlap as is shown in Figure 10. However, if $x(t)$ were to be sampled below the Nyquist rate to produce the signal $x_2[n]$, overlapping would occur between the spectral copies in the DTFT results ($DTFT\{x_2[n]\}$). This overlapping is shown in Figure 11. Because the results of the DFT are merely a sampled version of the results of the DTFT, this example is also applicable to the DFT and, therefore, FFT algorithms.

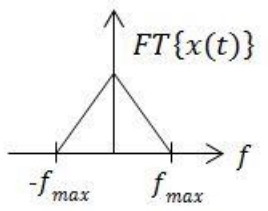

Figure 9: Fourier Transform Frequency Envelope

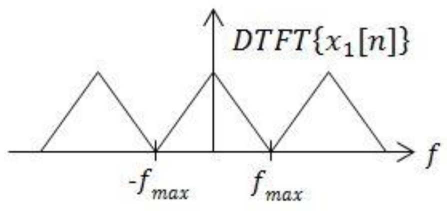

Figure 10: Transformed Signal Sampled at the Nyquist Rate

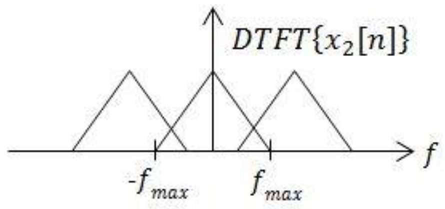

Figure 11: Transformed Signal Sampled at Below the Nyquist Rate

Aliasing has the effect of corrupting frequency components in the base copy of the signal spectrum with the addition of higher frequencies from adjacent spectral copies that are beyond the maximum frequency that can be represented by the FFT operation. Often, it is impossible to determine beforehand what the maximum frequency content of a signal will be. In such cases, in order to prevent aliasing, the continuous-time domain signal must be processed with a low-pass filter to remove any frequency components that are above the maximum allowed frequency for a given sampling rate. This process is called anti-aliasing [7]. It is important to note that since aliasing occurs at the time of sampling, this filtering must occur in the continuous-time domain for maximum effect.

If $x(t)$ were to be passed through an anti-aliasing filter, the maximum frequency of the result ($y(t)$) would now be the cutoff frequency of the filter (f_c). A representation of the resulting signal spectrum is shown in Figure 12. Because the maximum frequency content of the signal has been lowered, the Nyquist rate for the signal is also lowered. Correct selection of f_c can eliminate aliasing altogether. This scenario is represented in Figure 13.

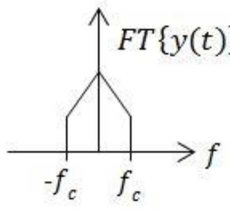

Figure 12: Low-pass Filtered Signal Spectrum

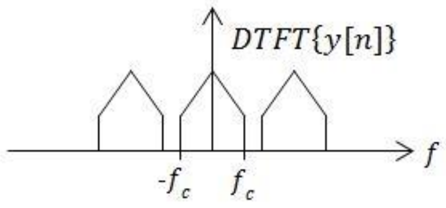

Figure 13: Successful Anti-Aliasing

The final issue of concern with regards to the FFT operation is signal window periodicity. The DFT, and therefore the FFT, take a finite number of points as time-domain input. However, the concept that the FFT is built on requires the time domain signal to be periodic with infinite duration. This is dealt with mathematically by making the assumption that the window in the time domain signal is repeated to infinite time. In this way the signal is made to be, in concept, a periodic signal with infinite duration [7]. The period of this conceptual signal is the window width, T_N. It is worth noting at this point that the period of this conceptual signal is related to the bin size of the results by (4).

Because the measured samples are conceptually repeated, end to end, for an infinite duration, it is important to make sure that there is a whole number of fundamental periods of the signal contained in a single window width. If, for example, a sine wave with a period of 20 ms is operated on by an FFT algorithm with a window width of only 10 ms, the results would not be consistent with the Fourier transform of the original continuous-time domain signal. In this example, the window width is only half of a period. Therefore, the conceptual signal made up of periodically replicated sample windows contains discontinuities and clearly differs from the original continuous-time signal. This effect is shown in Figure 14.

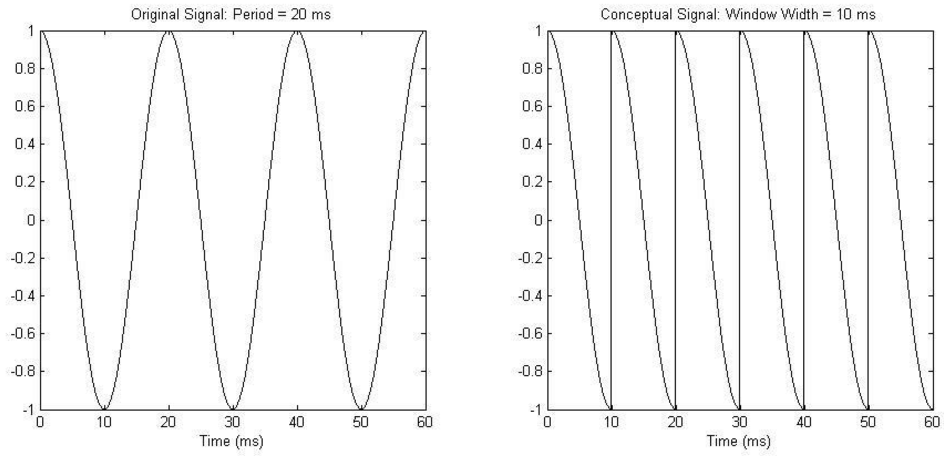

Figure 14: Signal Window Periodicity Example

4.3: Sample Rate Conversion

The sampling frequency used to acquire a discrete-time domain signal from a continuous-time domain signal affects both frequency bleeding and aliasing and is further limited by the need to have a power of two number of samples per window width. This frequency, therefore, must be chosen with care. However, sampling frequency is often dictated by the limited capabilities of the sampling hardware. Digital systems contain one or more oscillators which provide electrical pulses at regular intervals. This pulsing signal is often referred to as the clock signal [8].

The timing of all operations in the digital system is tied inextricably to the speed of the system's oscillator. No operation can occur faster than the frequency of the clock signal. Also, since these clock pulses serve as the triggers for commands to execute in digital systems, no periodic task can operate at a frequency that is not a scaled version of the clock frequency. For example, if a given clock signal had a frequency of 1 MHz, then periodic tasks could be performed at 1 MHz, 500 kHz, 333.33 kHz, and so on. Any frequency that could be found by dividing the clock frequency by an integer would be attainable. However, other frequencies, like 800 kHz for example, would not be possible using a 1 MHz oscillator. Scaling a clock frequency down by dividing it by an integer is known as prescaling [9].

In a real digital system, the sampling frequency of an ADC must be selected from the list of available operating frequencies related to the system's oscillator. Sometimes, this set of available frequencies is too limiting and some method of virtually acquiring a sampling frequency outside this set is desired. This process is called sample rate conversion.

One method of performing sample rate conversion is linear interpolation. This method functions by approximating the original continuous-time domain signal by connecting all of the samples in the discrete-time signal by straight lines. Once the original signal is approximated, new points are sampled in software, virtually approximating any desired sampling frequency. An example of this process is shown in Figure 15.

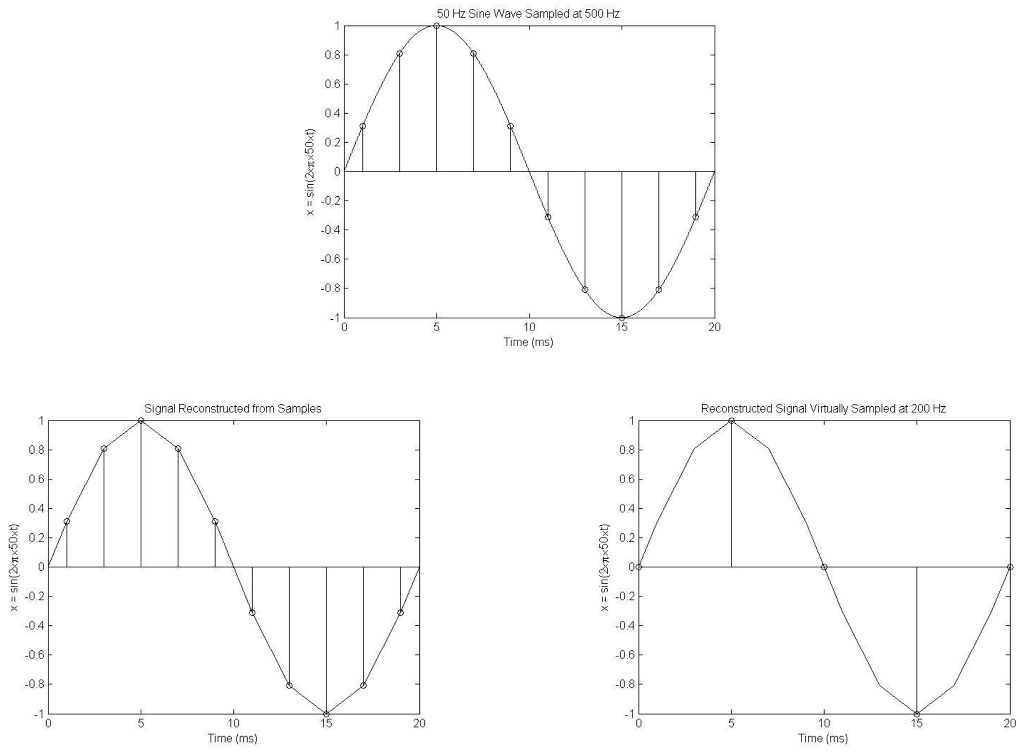

Figure 15: Sample Rate Conversion by Linear Interpolation

This approach does have limitations. The primary limitation is that the reconstructed version of the original signal is not perfect. Because of this, some error is introduced into the new virtually sampled sequence. In order to limit the effects of this error, the original sampling frequency should be greater than the virtual sampling frequency that is desired, which in turn, should be at least as fast as the Nyquist rate.

Chapter 5: Sample Application

Now that the necessary concepts of signal processing are understood, it is possible to proceed with implementing a harmonic measurement scheme using a PC sound card as a voltage measurement device. Harmonic measurement is a broad category which encompasses everything to do with measuring and calculating quantities related to harmonic components present in power signals. In relation to topics such as this one, it is important to carefully define what is measured and how it is calculated. One source of these definitions can be found in IEC standard number 61000-4-7 [10]. This standard clearly lays out what exactly is meant by harmonic measurement.

5.1: IEC 61000-4-7

The IEC definition of harmonic measurement applies to measuring voltage quantities and is therefore applicable to the PC sound card measurement platform. The harmonic measurement process according to the IEC standard [10] is shown in Figure 16. The preprocessing block refers to any operations that must be performed on the continuous-time domain signal before sampling. Anti-aliasing is an example of an operation that would occur in this step. However, designing and building an analog low-pass filter was deemed outside the scope of this project and, therefore, will not be discussed.

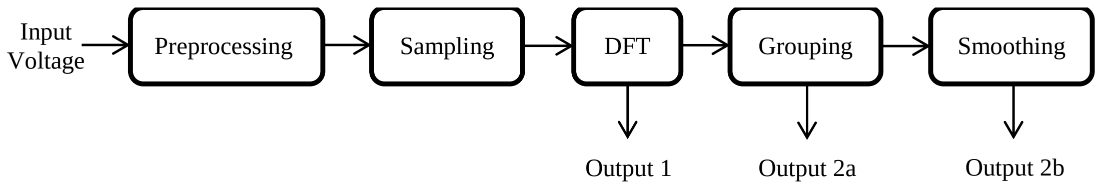

Figure 16: Harmonic Measurement Process

Another issue that would be handled in preprocessing is that the voltage signal that is to be measured for this example application is a 120 Vrms signal from a residential outlet. Because the maximum input amplitude of the mic input jack was determined to be 130 mV, some preprocessing of the signal is necessary to reduce the voltage to an acceptable level. The circuit used for this preprocessing is shown in Figure 17.

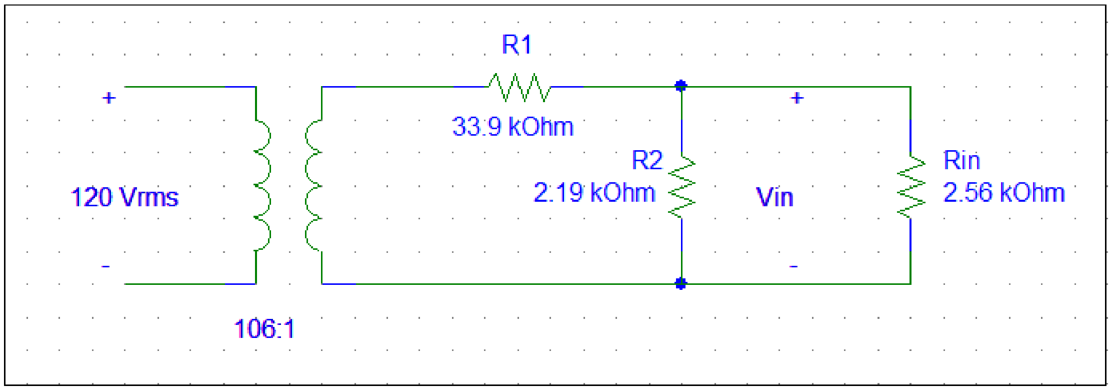

Figure 17: Preprocessing Circuit Diagram

The first step in reducing the input voltage was to connect a transformer. The transformer that was used was a PARALINE SSC5-4/480. Though the rated turns ratio was 120:1, experimentation revealed that the effective turns ratio was closer to 106:1.

The transformer drops the input voltage to approximately 1 Vrms. Clearly this is still too high to be used as input to the mic jack. Therefore, a voltage divider network was used to reduce the voltage a little further. Resistors R1 and R2 in Figure 17 make up this network.

Finally, the input resistance of the mic jack was measured using a digital multi-meter (DMM) while the device was recording. This measured value is shown as Rin in Figure 17. This approximate model of the preprocessing circuit leads to calculating an input voltage of approximately 54 mV peak for a supply voltage of 120 Vrms. This value is below the maximum recordable voltage for the mic input jack of 130 mV peak. Furthermore, using the previously established conversion chart from input voltage to recorded floating point value along with the transformer and voltage divider network, 120 Vrms roughly maps to a peak float value of 0.399.

The sampling block refers to the process of creating a discrete-time domain signal from the preprocessed continuous-time domain signal. This operation is accomplished through the use of an ADC. In the specific case of the PC sound card measurement platform, this operation is performed by the sound card hardware in concert with the driver software.

Additionally, any sample rate conversion that is required is performed in the sampling step. Because an FFT routine is used in later steps, it is important to collect a power-of-two number of samples in every sample window. The IEC standard dictates that the window width be 200 ms [10]. This length of time corresponds to ten cycles in a 50 Hz power system and twelve cycles in a 60 Hz power system.

As such, a virtual sampling rate of 10.24 kHz was selected to produce 2048 samples in every 200 ms window. Assuming that the highest frequency of interest in the recorded signals is the 50th harmonic of a 60 Hz power system, 3 kHz, the Nyquist rate for this measurement setup is 6 kHz. The selected virtual sampling rate is faster than the

Nyquist rate and is slower than the hardware sampling rate and should, therefore, result in minimal error.

The DFT block performs the discrete-time Fourier transform on the sampled sequence. The frequency domain results are referred to as Output 1 and are passed to the next block for further processing. The individual quantities that are calculated to form Output 1 are a_k, b_k, and $Y_{C,k}$. The first two quantities are the coefficients in the sine-cosine form of the Fourier series as shown in (2) in Chapter 4. The $Y_{C,k}$ term is the RMS value of the coefficients in the single sine form of the Fourier series as shown in (1) in Chapter 4. The equations for calculating these quantities are shown in (5), (6), and (7) [10]. In these equations, T_N is the window width for the DFT operation in seconds, ω_1 is the fundamental frequency of the measured signal in radians per second, N is the number of fundamental periods in one window width, and k is the spectral order.

$$a_k = \frac{2}{T_N}\int_0^{T_N} f(t)\cos\left(\frac{k}{N}\omega_1 t\right)dt \tag{5}$$

$$b_k = \frac{2}{T_N}\int_0^{T_N} f(t)\sin\left(\frac{k}{N}\omega_1 t\right)dt \tag{6}$$

$$Y_{C,k} = \frac{c_k}{\sqrt{2}} = \frac{\sqrt{a_k^2 + b_k^2}}{\sqrt{2}} = \sqrt{\frac{a_k^2 + b_k^2}{2}} \tag{7}$$

The grouping block in Figure 16 forms harmonic groups from the spectral components immediately surrounding the harmonic frequency components. This operation is performed because of the frequency bleeding phenomenon. If the spectral content of the measured signal does not fall precisely on the expected harmonic value, then the energy at that harmonic frequency will bleed into the surrounding frequency bins in the DFT result. In order to catch potentially bleeding energy, the spectral components surrounding the harmonic components are summed into harmonic groups. These groups make up Output 2a and are passed to the next block for further processing. The harmonic group quantities are referred to as $Y_{g,h}$. The equation used to calculate these quantities is shown in (8) [10]. In this equation, the $Y_{C,k}$ terms are the results of equation (7) and h is the harmonic order.

$$Y_{g,h}^2 = \frac{1}{2} Y_{C,(N\times h)-N/2}^2 + \sum_{k=(-N/2)+1}^{(N/2)-1} Y_{C,(N\times h)+k}^2 + \frac{1}{2} Y_{C,(N\times h)+N/2}^2 \qquad (8)$$

The smoothing block in Figure 16 passes the results of the grouping block over time through a low-pass filter. This filtering removes any sudden disturbances from the grouped harmonic values over time and results in a smooth plot of harmonic values over time. This smoothed output makes up Output 2b and is referred to as $Y_{og,h}$. A diagram depicting the low-pass filter used in the smoothing operation is shown in Figure 18 [10]. The α and β values are prescribed by the IEC standard and are shown in Table 1. The z^{-1} operation refers to a delay of one window width. Output 2b is then further processed and compared with harmonic limits standards to determine if the measured voltage signal is in compliance.

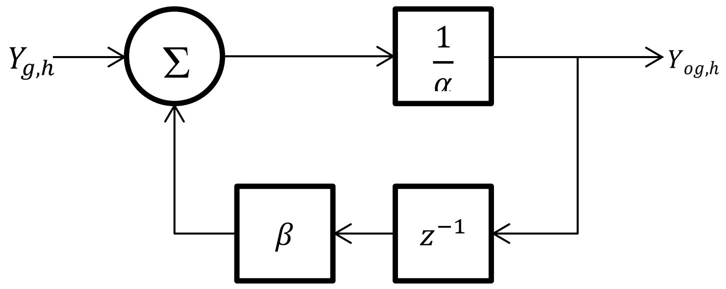

Figure 18: Smoothing Filter Diagram

Table 1: Smoothing Filter Values

α	8.012
β	7.012

Because these harmonic measurements are taken over an extended period of time, the timing of the calculations is important. The smallest unit of time relevant to these calculations is the 200 ms window width used by the DFT operation. Furthermore, the IEC standard dictates that the 200 ms values be averaged using the root mean square method every 3 s. These 3 s values are then averaged using the same method every 10 min. There are, therefore, new values every 200 ms, 3 s, and 10 min.

5.2: Harmonic Calculations

Equations (5) and (6) are in a continuous-time domain form. In order to implement these calculations using a computer to find values for a_k and b_k, they must be converted to a discrete-time domain form. This was accomplished by approximating the integral using trapezoidal integration to convert the integral to a sum.

Integration is finding the area under a curve. Trapezoidal integration approximates a continuous curve from discrete points by assuming that each point is connected to the next by a straight line. This straight line forms the top of a trapezoid between each pair of adjacent points. The area of each trapezoid is calculated and summed to approximate the integral. The resulting equations for a_k and b_k are shown in (9) and (10). The variable M represents the total number of points in the sequence being used and f_1 represents the fundamental frequency of the measured signal in Hertz.

$$a_k = \frac{2}{T_N}\int_0^{T_N} f(t)\cos\left(\frac{k}{N}\omega_1 t\right)dt$$

$$a_k = \frac{2}{T_N}\sum_{m=1}^{M-1}\left\{\frac{f[m] * \cos\left(\frac{\omega_1 T_N}{NM} km\right) - f[m-1] * \cos\left(\frac{\omega_1 T_N}{NM} km\right)}{2}\right\} * \left\{\frac{mT_N}{M} - \frac{(m-1)T_N}{M}\right\}$$

$$a_k = \frac{2}{T_N}\sum_{m=1}^{M-1}\left\{f[m] * \cos\left(\frac{2\pi f_1 T_N}{f_1 T_N M} km\right) - f[m-1] * \cos\left(\frac{2\pi f_1 T_N}{f_1 T_N M} km\right)\right\} * \frac{T_N}{2M}$$

$$a_k = \frac{1}{M}\sum_{m=1}^{M-1}\left\{f[m] * \cos\left(\frac{2\pi}{M} km\right) - f[m-1] * \cos\left(\frac{2\pi}{M} km\right)\right\} \qquad (9)$$

$$b_k = \frac{1}{M}\sum_{m=1}^{M-1}\left\{f[m] * \sin\left(\frac{2\pi}{M} km\right) - f[m-1] * \sin\left(\frac{2\pi}{M} km\right)\right\} \qquad (10)$$

The results shown in (9) and (10) are computationally intensive. For each a_k or b_k value, every single point in the source sequence must be used. Experimentation revealed that this level of computation could not be achieved on the test machine as a real-time calculation. It took longer to calculate all of the a_k and $b_k$values than it took to collect a buffer's worth of time domain data. This consistently led to buffer overflow errors in the program, causing it to cease functioning within seconds.

Alternatively, the a_k and $b_k$values could be bypassed by calculating c_k directly using an FFT algorithm. The $Y_{C,k}$ values could then be calculated as before, and the process would then continue without disruption. The new set of output variables would be identical to the previous set with a_k and b_k removed. Experimentation revealed that with a buffer length of three seconds, this alternative calculation method could be sustained in real-time indefinitely.

5.3: Experimental Results

Once all of the real-time calculations were set up, an experiment demonstrating the capabilities of the measurement platform was executed. The voltage at a residential 120 Vrms, 60 Hz outlet was measured for a period of one week. Frequency components up to the 50th harmonic were calculated. All of the 3 s and 10 min values were recorded in files for later analysis.

Typical spectral content for the 3 s values as a percentage of the fundamental component is shown in Figure 19. Typical 10 min values are shown in Figure 20. No significant spectral content was measured beyond the 7th harmonic. Smoothed, 10 min values for the 3rd, 5th, and 7th harmonics as a percentage of the fundamental component are shown plotted over the full week in Figure 21.

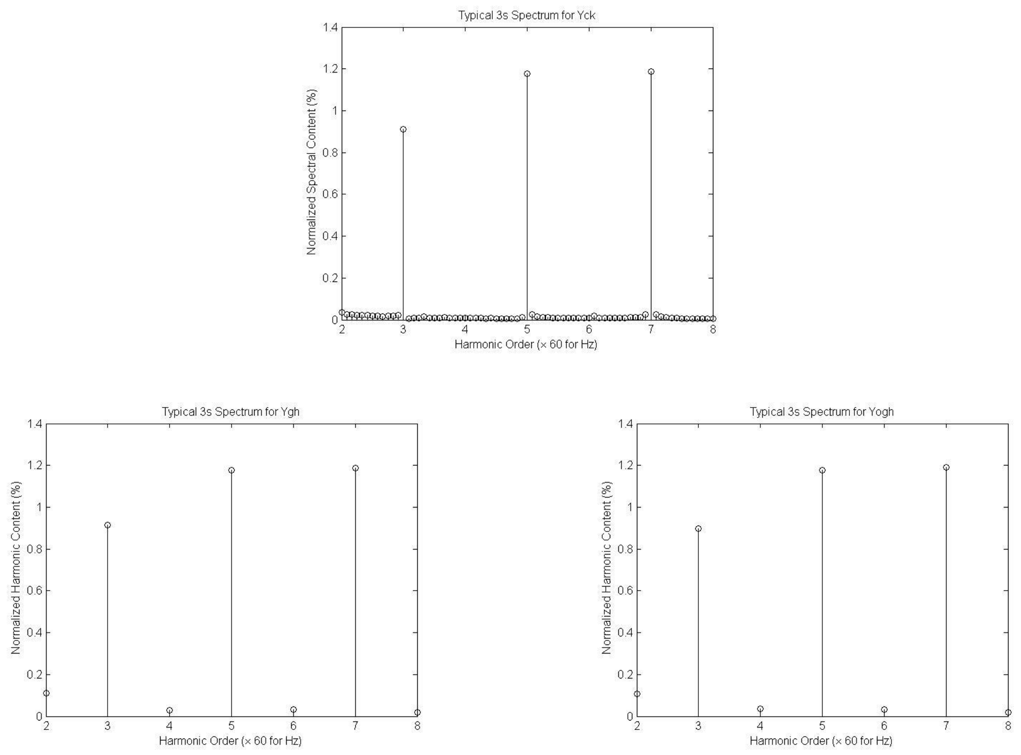

Figure 19: Typical Three Second Spectra

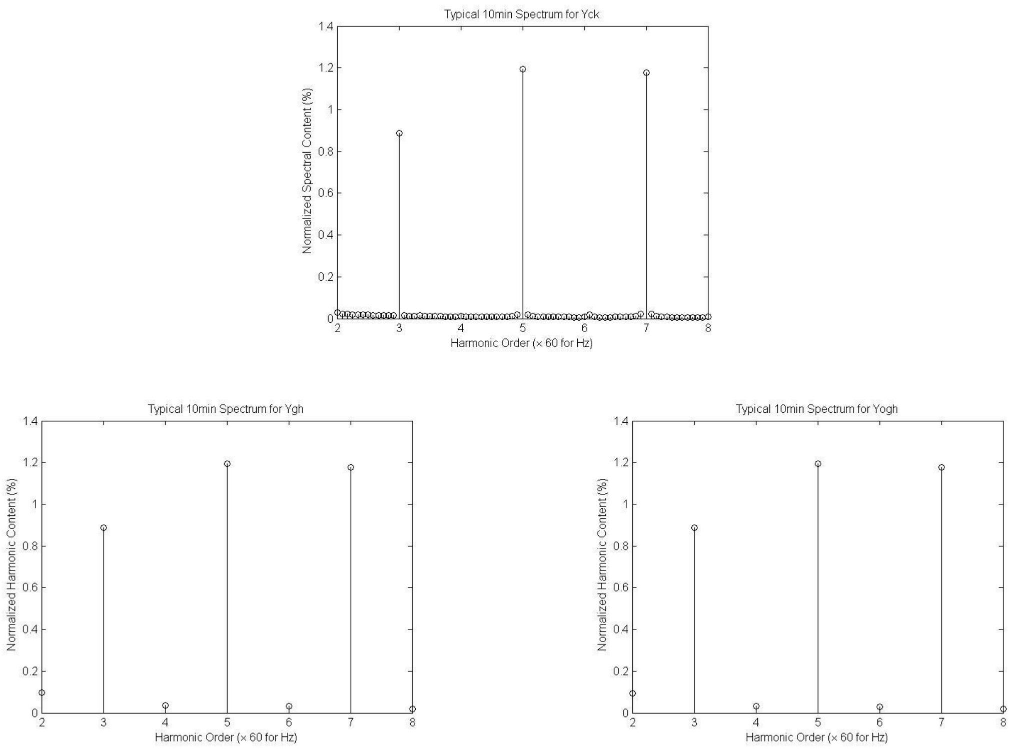

Figure 20: Typical Ten Minute Spectra

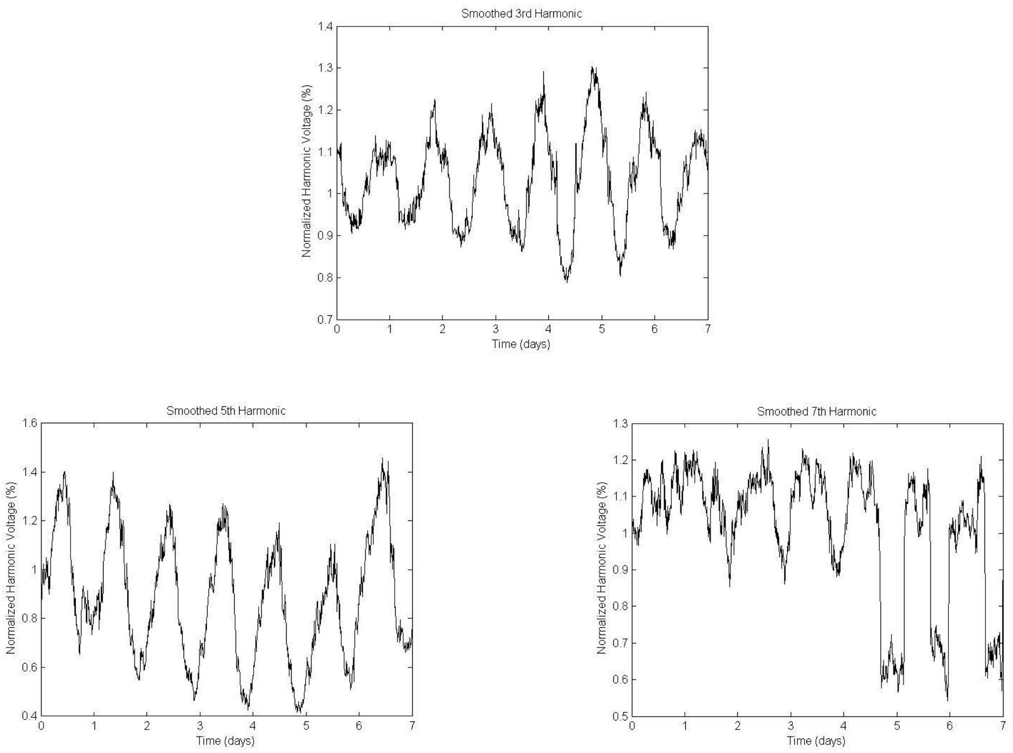

Figure 21: Smoothed Values for 3rd, 5th, and 7th Harmonics

The plots in Figures 19 through 21 are scaled based on the recorded value for the fundamental component. However, the time domain plot in Figure 22 attempts to map the recorded values back to the outlet voltage. This mapping requires some calibration in the program. In this case, the calibration was performed using circuit analysis based on the circuit values shown in Figure 17 and the transfer characteristics shown in Figure 3.

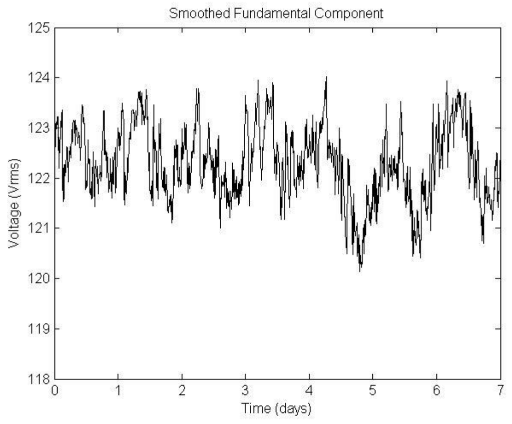

Figure 22: Recorded 60 Hz Voltage

At a later time than when the experiment was run, independent measurements were taken of the same outlet voltage using a DMM. These measurements were taken periodically over several hours and ranged from 120.9 Vrms to 123.0 Vrms. These values seem to corroborate the values shown in Figure 22. However, without extensive testing, it is impossible to verify that the harmonic measurement results are accurate. No claim is being made to that effect. Instead, these results clearly demonstrate that this design for a voltage measurement platform using a PC sound card is capable of meeting all usability requirements.

Chapter 6: Conclusions

The program written as the core of this project was able to communicate with and help quantify the electrical characteristics of an example PC sound card. Use of the PortAudio library allowed for a simple interface between C++ code and the sound card hardware, regardless of driver specifics. Overall, the usability requirements as set out in the Introduction were met by the tested sound card.

First, it was demonstrated that the sound card is capable of continuous recording. Once the appropriate API was selected for communication with the endpoint device, no data was dropped or skipped. The only tradeoff required to assure this continuity is to implement sample rate conversion in the application as part of data processing.

Second, it was demonstrated that the test system was capable of performing rigorous calculations while recording data in real time. Not all variables of interest could be calculated. The a_k and b_k values required by the IEC standard were too computationally intensive to calculate in real time. However, all of the remaining values were calculated without any loss of data.

Finally, all of the continuous recording and strenuous real time calculations were sustainable over an extended period of time. As already stated, the sample application experiment ran without interruption for a whole week. During this time, the program did not halt for any reason, nor were any samples missed or calculations left incomplete.

Now that the feasibility of this approach has been established, further research is warranted. In the harmonic measurement application used as an example in this experiment, more work is needed to establish the frequency response of the preprocessing circuit shown in Figure 17. The assumption was made that the input impedance of the mic in port was purely resistive; this is almost certainly not the case. Additionally, the transformer used in the circuit behaves differently depending on the frequency of the signal passed through it. The realistic behavior of the components in the preprocessing circuit will produce an effect on the overall frequency response of the system. Work could also be done to make the calculation code more efficient to allow the computation of the a_k and b_k values required by IEC standard 61000-4-7. Furthermore, the final results of the experimental measurements must be independently verified before the voltage measurement platform can be used as a trusted DAQ.

Other applications can be approached as well. Any process which requires voltage measurements to be processed by a computer can and should be attempted using this voltage measurement platform. For each new application, new code will have to be developed and the results independently verified by equipment which already exists for the desired purpose.

References

[1] Claus Riethmüller, “Windows Driver API Basics,” ST Audio Knowledge Base, <http://www.staudio.de/kb/english/drivers/>, January 2003, accessed June 2013.

[2] PortAudio, “PortAudio API Overview,” <http://portaudio.com/docs/v19-doxydocs/api_overview.html>, accessed July 2013.

[3] Mark Heath, “Sound Code: What’s Up With WASAPI?,” <http://mark-dot-net.blogspot.com/2008/06/what-up-with-wasapi.html>, June 2008, accessed June 2014.

[4] National Instruments, *NI Educational Laboratory Virtual Instrumentation Suite II Series (NI ELVIS II Series) User Manual*, January 2009.

[5] Edward W. Kamen and Bonnie S. Heck, *Fundamentals of Signals and Systems Using the Web and MATLAB*, 3rd ed., Pearson Education 2007.

[6] National Instruments, “Zero Padding Does Not Buy Spectral Resolution,” <http://www.ni.com/white-paper/4880/en>, September 2006, accessed July 2013.

[7] Alan V. Oppenheim and Ronald W. Schafer, *Discrete-Time Signal Processing*, 3rd ed., Prentice Hall 2010.

[8] Victor P. Nelson, H. Troy Nagle, Bill D. Carroll, and J. David Irwin, *Digital Logic Circuit Analysis and Design*, Prentice Hall 1995.

[9] John Hung and Victor Nelson, “Embedded Systems Design Lab,” Lecture, Fall 2010.

[10] IEC, *Amendment to IEC 61000-4-7 Ed.2: Electromagnetic compatibility (EMC) : Testing and measurement techniques – General guide on harmonics and interharmonics measurements and instrumentation, for power supply systems and equipment connected thereto,* March 2008.