SonicData: An Open-Source Instrument Data Acquisition Program

Benjamin Stauffer

March 16, 2007

Joseph A. Counsil (Department of Chemistry)

Bruce McMillin (Department of Computer Science)

University of Missouri-Rolla

Rolla, MO 65401

Abstract

A low-cost voltage-to-frequency interface can be used to collect and process data from instrumentation by means of a sound-card input to a personal computer. A program was designed to read, analyze, process, and generate reports from the resulting data stream. The program design was based upon a list of requirements from the user, including parameters specifying data rate and range, real-time presentation, data analysis, report generation, and the user interface. The program was written in C++ using open source libraries. Program operation is adaptable through use of a configuration file. The source code was designed to be easily adapted for other applications. Users can save the data in a graphical, plain text, or spreadsheet format.

SonicData: An Open-Source Instrument Data Acquisition Program

The usefulness of old instrumentation in the UMR Chemistry department is limited by the lack of modern computing capabilities. The department's instrumentation laboratory contains a combination of old and new instrumentation, some of the newer being computerized. However, the older instrumentation presents its output on paper-based chart recorders, precluding the ability to save data in computer-based formats and manipulate data sets.

Options for updating the older instrumentation include adapting a preexisting interface to the devices or creating a new interface. Commercial computer interfaces are expensive, and specific to instrument type, thus requiring numerous non-uniform solutions. Therefore, this investigation seeks to determine the feasibility of providing a low-cost, easily adaptable interface for a variety of instruments, and to design such an interface.

Hardware Interface

A voltage-to-frequency converter to interface between laboratory instrumentation and a desktop computer was designed and built by Joseph Counsil of the UMR Chemistry department (see Figure 1). The interface generates a filtered square wave (see Figure 2). The fundamental frequency of the wave is directly proportional to the output voltage from the instrumentation, which ranges between 1 millivolt and 5 volts; however, due to the simplicity of the device, it may easily be adapted to support a wider range of voltages, allowing it to be used in a large number of applications.

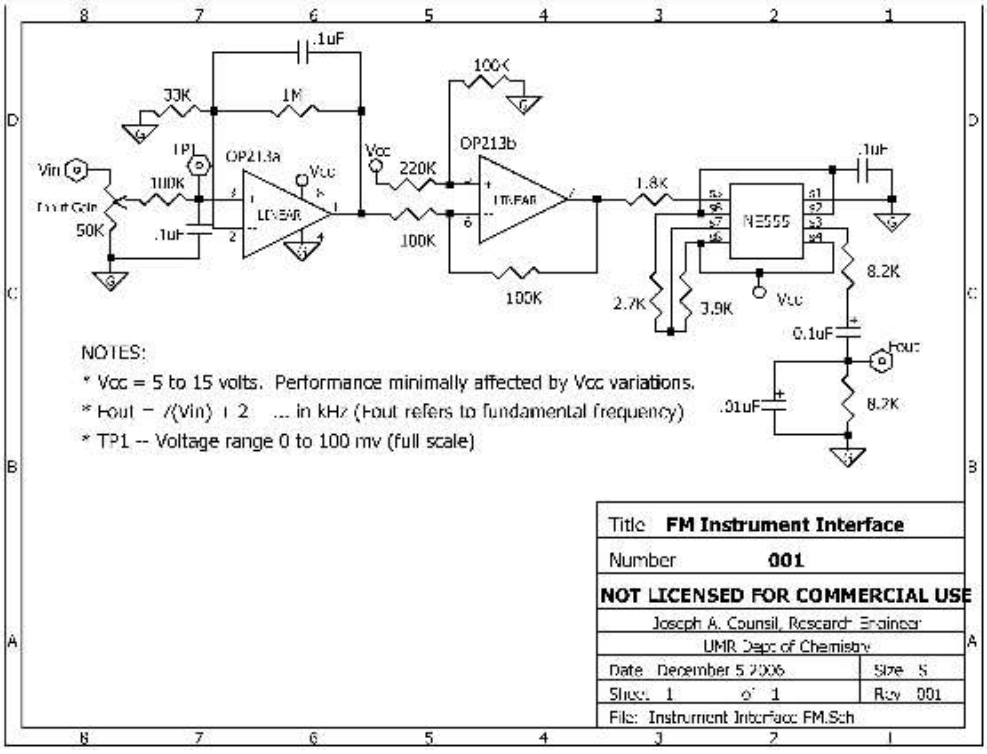

Figure 1. The schematic of the voltage-to-frequency converter.

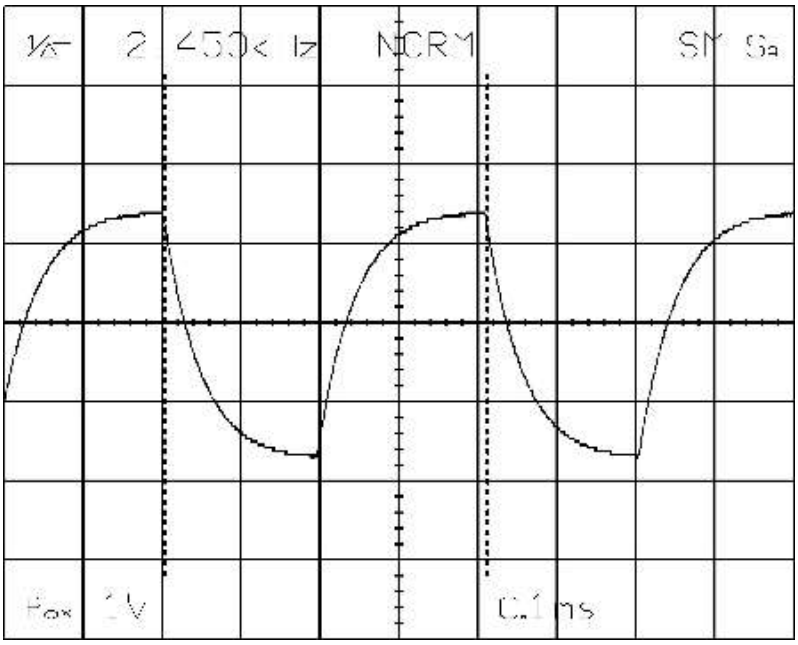

Figure 2. An oscilloscope output showing the filtered square wave generated by the voltage-to-frequency converter.

Due to the fact that the device is generating a filtered square wave and not a sine wave, there are harmonic frequencies being generated, as shown in the Spectran[1] screenshot in Figure 3. Therefore, Digital Signal Processing (DSP) must be performed to filter the harmonics, thus isolating the remaining fundamental frequency from which data points may be determined.

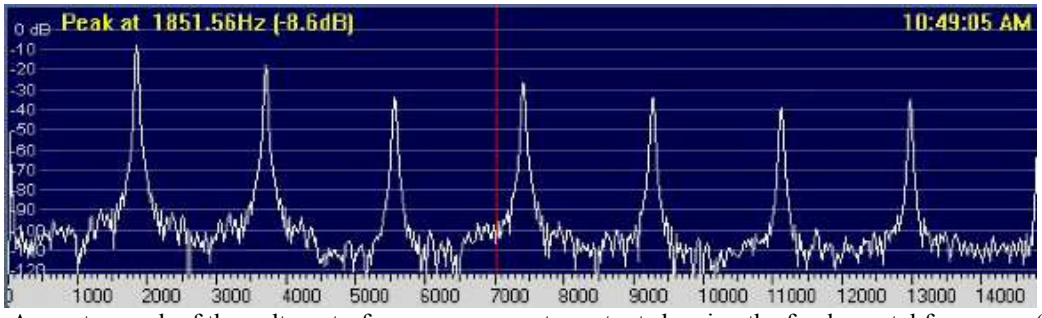

Figure 3. A spectrograph of the voltage-to-frequency converter output showing the fundamental frequency (the highest peak) and harmonics.

The device has been demonstrated to work successfully using a computer sound card as an analog-to-digital data-acquisition interface. The remaining need is for a software application to interface with the device to collect and process the data, generate reports, and allow the user to save the data in a useful file format.

Program Requirements

The requirements of this program were determined based upon the instrumentation needs. They include the following:

1. As the hardware will generate a frequency oscillator between 500 and 3000 Hz, the software must be able to correctly detect and filter frequencies within this range.

2. The accuracy of the determined frequency must be three significant digits.
3. An experiment may be run for a length of time predetermined by the user. The user must be able to extend the time or end the run early as desired.
4. Raw sampling rates must range from 100 samples per second to 1 sample per hour.
5. The program must give the user an option to calculate and store the average of every N samples, where N is specified by the user.
6. The program must be compatible with Windows 98 and newer. Compatibility with Linux is preferred. Compatibility with Windows 95 is desirable but not required.
7. The software must include:
    a. An easily readable and modifiable configuration file, allowing the program to be configured for a number of desirable applications.
    b. A configuration menu allowing users to modify the options given in the configuration file.
    c. A method of preventing end users from modifying certain settings as determined by the system administrator.
    d. Graphical real-time data presentation during data acquisition.
    e. An end-of-run graph displayed following acquisition, automatically scaled, with peaks in the collected data marked and labeled.
    f. An end-of-run report must present the time of each peak in the data and the percentage of the total area under each peak.
    g. The ability for the user to zoom in on a data subset following collection and process it separately as desired.
    h. The ability for the user to save collection results as an Excel spreadsheet or as a text document with a configurable delimiter.
8. The program must be written in a maintainable fashion, with appropriate comments in the code and documentation fully explaining its operation.
9. The source code of the program will be released under the GNU Public License; as such, it can only use libraries compatible with this license.

Program Development

SonicData was developed in C++ using the MinGW[2] C++ compiler for Windows. Using MinGW was preferable as it is a port of the GNU[3] compilers commonly used on Linux systems, allowing easier cross-platform development (see requirement 6). The Allegro[4] graphics library was used for all graphical output. The PortAudio[5] library was used for audio input. These libraries were chosen because they are available for both Windows and Linux platforms (see requirement 6), and they are both compatible with the GNU Public License (see requirement 9).

The program builds on the functionality of another program called DansTuner[6], a software tuner for musical instruments, which was discovered during pre-development research. Most of the code involving audio input and frequency calculation was borrowed from DansTuner, but it was modified to meet the requirements of the user. As DansTuner was released under the GNU Public License, the source code for a derivative project like SonicData must also remain freely available under this license.

One of the most important features of the program is an easily modifiable configuration file which allows the user to tailor the program for a wide range of instruments with changes to sampling rate and/or frequency range. The configuration file is an ASCII text file and is arranged in an easily readable format. At the system administrator's discretion, certain parameters may be protected from user modification.

A number of challenges were discovered in the process of writing the program. Among the first of these was learning how to compile and use the PortAudio and Allegro libraries, which was resolved using documentation found on the Internet. While testing a prototype of the program with an early prototype of the voltage-to-frequency converter, it was discovered that changing the operating system volume settings and modifying the filtering method used by DansTuner yielded far more reliable data. Early in the program development it was suggested that multithreading be used to ensure that samples would be collected at correct intervals regardless of the ability of the system to output the data to the screen or to a file in real time. Later on, it was discovered that multithreading in MinGW would be more difficult to implement as it is not consistent with the POSIX style threads used by Linux-based systems. With the user's approval, it was decided to test the program without multithreading, and the performance was determined to be satisfactory.

Program Operation

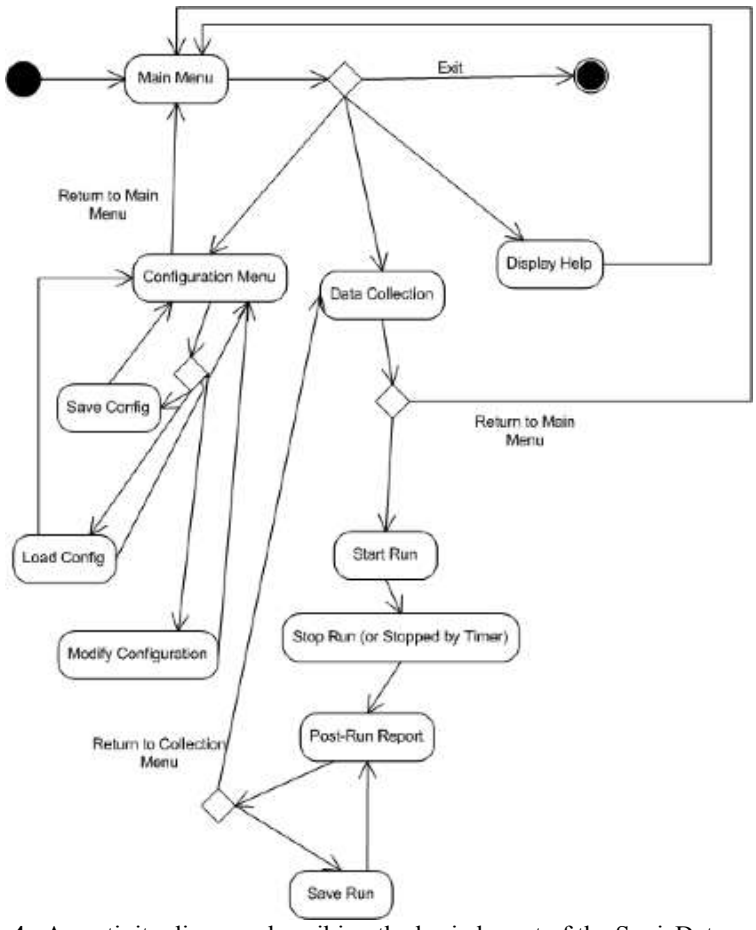

Figure 4. An activity diagram describing the basic layout of the SonicData program.

When the program is run, configuration data is loaded from the configuration file and stored in a struct, and the main menu is displayed. Each menu button is associated with a separate function in the main implementation file. A pointer to the configuration struct is passed to all menu functions.

In the configuration menu, the user may alter the any settings that have not been locked by the system administrator; the variables within the configuration struct are modified to reflect the new desired values, and the modified settings may be saved to a configuration file.

Upon calling the data collection routine (see Figure 5), a PortAudio recording stream is initialized. The collection routine draws a graph on the screen and then waits for the user to click the "Start Collection" button to begin the run. Once clicked, the program begins a loop which will terminate once the user clicks "Stop Collection" or, in the case of a timed run, the run time is completed. Within the loop, a function called audioSoFar is repeatedly called. The audioSoFar function calls two functions, Autocorrelation and bestPeak. The Autocorrelation function windows the data using a Bartlett function, which was found to yield more accurate results than the Hamming function used by DansTuner. It then processes the audio input using a Fast Fourier Transform function to find the peaks in the sample. The bestPeak function returns the tallest of the peaks, which represents the fundamental frequency of the audio input. Program control returns to the collection loop. If the user desires to average a given number of points, the new value is added to a total which will be averaged and graphed once the desired number of points is collected. Otherwise, the frequency is directly added as a point on the graph. If the previous point is determined to represent a peak in the graph based on the current point, it is marked accordingly. The loop sleeps for a given number of milliseconds before restarting. The length of the sleep partially determines how many points will be collected per second; other factors will include the time needed to complete the audio processing and the screen refreshing.

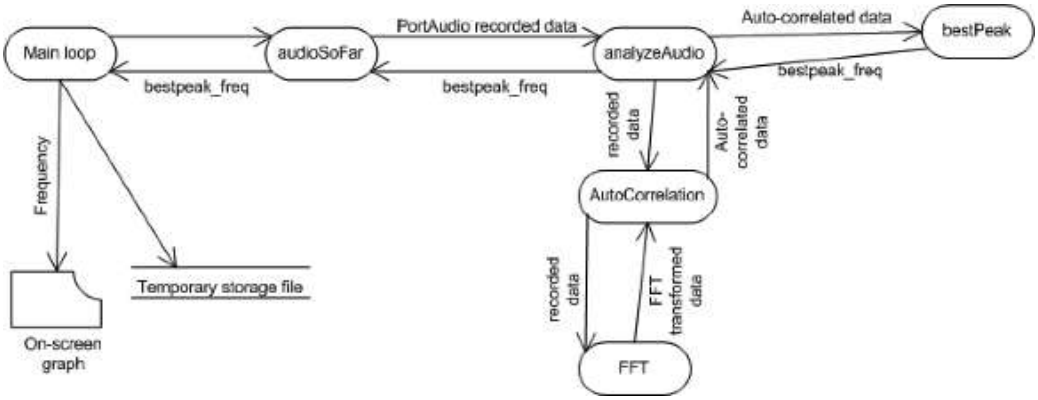

Figure 5. A data flow chart of the Data Collection routine.

Following a collection run, program control passes to a post-run analysis function, which integrates the area under each peak and generates a report stating the location and percentage of the total area under each peak. The user may choose to save the collection results, begin a new collection run, or return to the main menu.

Results

The program is currently in pre-alpha stages, and is set to be finished in April 2007. Data point graphing is functioning correctly and requires only a few modifications to be considered finalized. Once this is completed, the post-run report with peak integration will be finished.

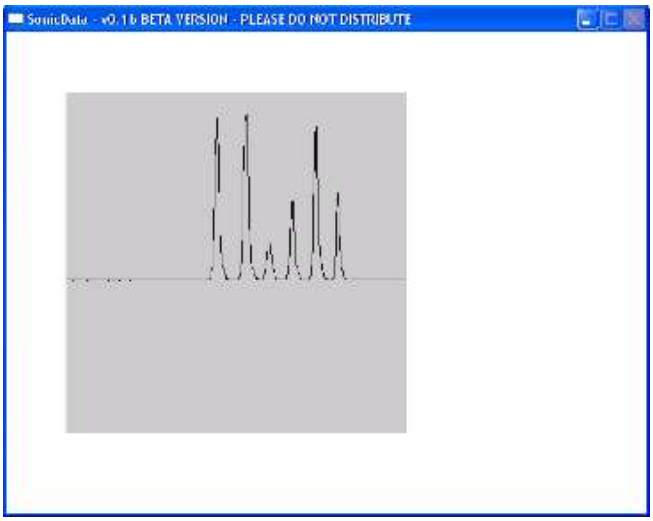

Figure 6. SonicData as of March 15, 2007. Data collection from the voltage-to-frequency converter is being shown. The voltage to the converter is being controlled by a temporary knob.

Discussion

In its current form, the program is not yet complete; however, its main functionality of collecting and displaying data is working as expected and is in near-final form. The accuracy of the collected data has been verified with other programs including Spectran. The program has been developed at no cost to the author.

Until this point, testing has been limited to Windows platforms. As the program was written to be cross-platform, however, it will be also thoroughly tested under Linux. Other platforms are also possible, such as the Apple Macintosh.

It is expected that the program will be hosted on SourceForge.net. An application to reserve http://sonicdata.sourceforge.net has been submitted and is under review at the time of this writing. SourceForge offers a number of beneficial tools for hosting open-source projects, including storage of previous versions of each file in the source code, bug tracking, and message boards for users who have questions regarding the program. The full source code of the program and the schematic for the voltage-to-frequency converter will be made available publicly under the GNU Public License.

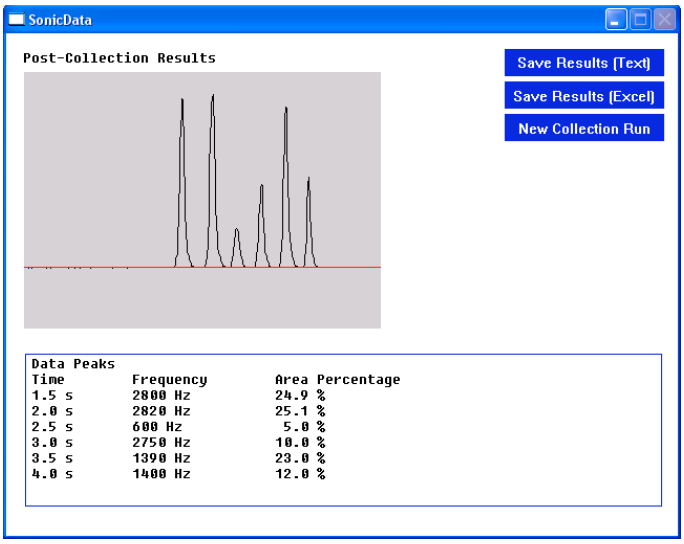

Figure 7. A simulated image showing the expected appearance of the post-collection screen. Note that all numbers in this image are simulated.

Acknowledgements

The author wishes to thank the following people for their contributions to the project: Joseph Counsil (research advisor), Dr. Bruce McMillin (Computer Science department advisor), Dan Frankowski (author of DansTuner), the authors and contributors of the PortAudio and Allegro libraries, and the authors of the MinGW compilers.

References

[1] Di Bene, Alberto. *Spectran Version 2*. (n.d.). Retrieved March 15, 2007, from http://digilander.libero.it/i2phd/spectran.html

[2] *MinGW – Minimalist GNU for Windows*. (2004). Retrieved March 15, 2007, from http://www.mingw.org

[3] *GNU's Not Unix! - Free Software, Free Society*. (2007). Retrieved March 15, 2007, from http://www.gnu.org

[4] *Allegro – A Game Programming Library*. (2007). Retrieved March 15, 2007, from http://alleg.sourceforge.net

[5] Burk, Phil. *PortAudio - Portable Cross-platform Audio API*. (n.d.). Retrieved March 15, 2007, from http://www.portaudio.com

[6] Frankowski, Dan. DansTuner. (n.d.). Retrieved March 15, 2007, from http://danstuner.sourceforge.net