Enhancing & Utilizing the DSP Toolkit of LabVIEW

Murat Tanyel
Geneva College, mtanyel@geneva.edu

Abstract - **Most Digital Signal Processing (DSP) courses rely heavily on MATLAB and/or C, representing the state of the art in textual programming, for their standard computer tools. We have argued, in previous papers, that whereas this environment may be efficient in manipulating equations, textual implementation of processes best described by block diagrams loses its intuitive substance and have provided examples of LabVIEW implementations that are better left graphical. The standard DSP toolkit of LabVIEW is aimed at the practicing engineer/scientist who needs to process acquired data to reach other ends in contrast to a student whose aim is to learn about signal processing. LabVIEW's DSP toolkit is rich with high level algorithms but needs to be enhanced in order to serve the pedagogical needs of students of DSP. While teaching this course at a previous institution, I developed many routines to complement the standard DSP toolkit as I tried to demonstrate basic concepts. Returning to teaching DSP at my new institution after a break of 6 years, I have found myself having to revise some of those old tools and have got excited about new possibilities that LabVIEW has to offer in this field. This paper will review my past experience and will focus on this additional toolkit that I have developed to make LabVIEW a better teaching tool in a DSP class. In particular, I will provide detailed descriptions of classroom activity that takes advantage of LabVIEW's sound capture and playback routines. The paper will conclude with an anecdotal description of students' reception of these activities.**

Index Terms – digital signal processing (DSP), digital audio effects, LabVIEW

INTRODUCTION

As computer applications have proliferated the electrical engineering curriculum, we observe that a number of applications have become widespread computer tools in electrical engineering textbooks. Spice and its derivatives, such as National Instrument's Multisim pervade courses that cover circuit analysis and electronics [1-7]; MATLAB and its derivative SIMULINK have become the standard computer tool for control systems [8-11], communication systems [12-14], digital signal processing (DSP) [15, 16]. The C programming language has replaced FORTRAN in the electrical engineering curriculum, as I have observed this transition from my undergraduate studies in the late seventies to graduate studies in the eighties. With the exception of SIMULINK and the graphical interface of Multisim, these different computer tools of the trade are text-based environments, as opposed to a newer breed of programming environments that take advantage of the more recent development of the graphical interface. The most ardent employer of this graphical programming environment has been National Instruments with their LabVIEW package that runs on a number of platforms, namely, MacOS, Windows, UNIX and Linux. A contender is Agilent VEE Pro, available on Windows and Vista. Another serious contender is SIMULINK with its textual roots on MATLAB.

I have contended that processes depicted by block diagrams in control systems, communication systems and DSP, are better candidates for simulation and/or realization in a graphical programming environment than in a textual environment [15]. When I taught DSP at my former institution, I used LabVIEW to implement my demonstrations and exercises.

This paper describes the recent experience I have had in using LabVIEW when I took over teaching DSP at my current institution after a long gap. I will first give a brief course description with the particular textbook used. I will then give an overview of DSP utilities of LabVIEW. I will proceed with examples which utilized the sound routines provided by LabVIEW. I will then convey the consensus of our small class on the effectiveness of LabVIEW in demonstrating DSP concepts, finishing with our concluding remarks.

THE OVERALL ENVIRONMENT: THE COURSE, THE FACILITIES AND THE TOOLS

The engineering department at Geneva College offers an ABET-accredited engineering major with civil, mechanical, electrical and computer engineering concentrations well as serving a non-accredited chemical engineering major provided by the Department of Chemistry. Digital Signal Processing is a senior level elective for electrical and computer engineering students. The textbook that I chose for this offering was Orfanidis' Introduction to Signal Processing [15]. This textbook was chosen because of my familiarity with it and its good balance between theory and practical applications as well as its many examples in C and/or MATLAB with the provision of code. The topics covered in this offering were: sampling and reconstruction, quantization, properties of discrete-time systems, FIR filtering and convolution, z-transforms, transfer functions, digital filter realizations (such as direct form, canonical form and cascade form, hardware realizations and circular buffers and quantization effects in digital filters), signal processing applications (digital waveform generators, digital audio effects), DFT/FFT algorithms, FIR digital filter design and IIR digital filter design.

ELE 440, Digital Signal Processing, is a three credit class which met for three 55-minute periods a week. The mode of instruction employed active learning in which students were required to read the topic of the day prior to coming to class and the class period was utilized to clear concepts, emphasize important points and to study practical applications.

Geneva College Engineering Department runs a number of its courses in tablet-enhanced mode, in which students are loaned tablet PCs. These computers are loaded with all the standard software for which the College in general and the Engineering Department in particular have licenses. ELE 440 is one of the tablet-enhanced courses. Therefore, National Instruments' LabVIEW is accessible to students in and outside of class. LabVIEW was chosen to supplement the DSP course for two main reasons: i) my prior experience with it [17-21] and ii) the recognition that it is a worthwhile programming environment that will complement the other packages, namely C++ and MATLAB, already introduced in other courses.

LabVIEW (short for Laboratory Virtual Instrumentation Engineering Workbench) is a graphical programming environment, based on the concept of data flow programming, particularly suited to test and measurement applications [22]. The three important components of such applications are data acquisition, data analysis and data visualization. LabVIEW offers an environment that covers these vital components. It is the combination of these specialized components and the data-flow programming paradigm that makes it attractive to scientists and engineers interested in quick test and measurement applications.

LabVIEW programs are called virtual instruments (VIs). The Signal Processing Library of the LabVIEW Full Development System (v. 8.5) contains VIs that are arranged in groups. The groups pertinent to the DSP course are listed below:

- **Waveform Generation**, containing VIs for generating different signals such as sine wave, square wave, various types of noises utilizing LabVIEW's waveform data structure.
- **Waveform Conditioning**, containing VIs for filtering, convolving and windowing waveforms.
- **Waveform Measurements**, containing VIs ranging from Basic RMS Value to Harmonic Distortion Analyzer, various types of Spectrum Analyzers on waveforms.
- **Signal Generation**, containing VIs for generating different signals such as a sine wave, a square wave, a chirp signal and white noise in simple, numerical arrays.
- **Signal Operation**, containing VIs for computing convolution, deconvolution, auto-correlation, cross-correlation, decimation and similar routines with numerical arrays.
- **Windows**, containing VIs for applying different kinds of windows such as Hanning, Hamming, Triangle, Blackman, Kaiser, on numerical arrays.
- **Filters** containing such procedures as Butterworth, Chebychev, Inverse Chebychev, Elliptic, Bessel high/low/bandpass/bandstop filters, advanced IIR filtering, advanced FIR filtering (windowed coefficients, Parks-McClellan algorithm, etc.) on numerical arrays.
- **Spectral Analysis**, containing VIs for computing the power spectrum, auto & cross power spectrum on numerical arrays.
- **Transforms**, containing VIs for computing the power spectrum, complex FFT, complex inverse FFT, fast Hilbert transform, inverse fast Hilbert transform and similar routines on numerical arrays.
- **Point by Point** many of the above routines performed on a single data point.

Employment of these readily-available VIs made many class demonstrations quick and intuitively easy to understand. The flexibility of the programming environment allowed us to write some more fundamental routines on our own, adding to LabVIEW's 'DSP toolkit' some specialized VIs for this class.

EXAMPLES UTILIZING THE SOUND VIS

One of the strengths of the textbook is the provision of many examples from digital music processing. LabVIEW's sound VIs, found under the category Functions / Programming / Graphics & Sound promised to be a good match to test the routines described in the textbook. The VIs under this category include:

- **Play Waveform**, which plays data in LabVIEW's waveform data structure,
- **Play Sound File**, which opens and plays .wav files,
- **Acquire Sound**, which samples sound from the host computer's standard microphone input,
- **Sound File Read**, **Sound File Write**, which read from and write onto .wav files.

All of these VIs utilize LabVIEW's waveform data structure, which includes the quantized sound data, sampling period and a time stamp bundled together. Following National Instruments' example, I will refer to data in waveform data structure as *Wavdata*. LabVIEW's Waveform Conditioning and Waveform Measurement VIs manipulate Wavdata. However, utilizing these canned VIs conceals the processing routines which I aim to teach students. In order to apply our own algorithms we developed in class, I developed VIs to work with the standard sound VIs and made them available on the course's Blackboard site. The following is a list of these VIs with brief descriptions:

- **GenerateSound.vi**: converts an ordinary numeric data array into LabVIEW's waveform data structure and plays it as a sound at the specified sampling rate (default is 8,000 samples/s).
- **WaveformToData.vi**: obtains the quantized sound data from LabVIEW's waveform data structure and stores in a numeric array. I will refer to the output of this VI as *raw sound data*.
- **NrOfChannels.vi**: determines the number of channels of sound in Wavdata, used as a subVI in WaveformToData.vi.
- **NrOfChannelsRaw.vi**: determines the number of channels of raw sound data.
- **StripChannel.vi**: strips the channel # indicated from raw sound data.

Equipped with these extra tools, we were able to implement and hear some of the examples provided in the textbook.

I. Sinusoidal Generators

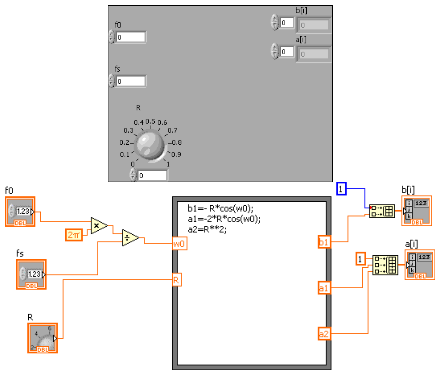
A filter with the transfer function

$$H(z) = \frac{R\sin\omega_0 z^{-1}}{1 - 2R\cos\omega_0 z^{-1} + R^2 z^{-2}} \qquad (1)$$

will generate an exponentially decaying sinusoid of frequency ω_0 for $0 < R < 1$ [15]. Similarly, a filter with the transfer function

$$H(z) = \frac{1 - R\cos\omega_0 z^{-1}}{1 - 2R\cos\omega_0 z^{-1} + R^2 z^{-2}} \qquad (2)$$

will generate a cosine waveform. Figure (1) depicts the front panel and block diagram of the subVI CosinusoidalCoeffGen.vi that we wrote as a class exercise. The inputs of this routine are f_0 (the frequency of the desired cosine), f_s (sampling frequency) and the parameter R. This subVI generates the numerator and denominator coefficients (forward and reverse coefficients) of Eq. (2) in a format that LabVIEW's IIR Filter routine calls for.



```
b1=- R*cos(w0);
a1=-2*R*cos(w0);
a2=R**2;
```

FIGURE 1
FRONT PANEL AND BLOCK DIAGRAM OF THE SUBVI COSINUSOIDALCOEFFGEN.VI.

II. DTMF Keypad

The digital touch tone phone, also known as the dual-tone multi-frequency (DTMF) transmitter/receiver is a typical application of sinusoidal generators. Each key on the keypad, when pressed, produces the sum of two tones. The signal, $y(n)$, generated by each key press may be expressed as [15]:

$$y(n) = \cos(\omega_L n) + \cos(\omega_H n) \qquad (3)$$

Each key is assigned its unique combination of lower and higher frequencies, ω_L and ω_H respectively. Table (1) summarizes the frequency assignments for a DTMF keypad.

TABLE I
DTMF KEYPAD FREQUENCY ASSIGNMENTS

		High Group			
		1209 Hz	1336 Hz	1477 Hz	1633 Hz
Low Group	697 Hz	1	2	3	A
	770 Hz	4	5	6	B
	852 Hz	7	8	9	C
	941 Hz	*	0	#	D

Figure (2) depicts the front panel of the KeypadSub.vi which simulates a DTMF keypad. This VI employs two of the VIs implemented for this offering of the course, namely GenerateSound.vi and CosinusoidalCoeffGen.vi.

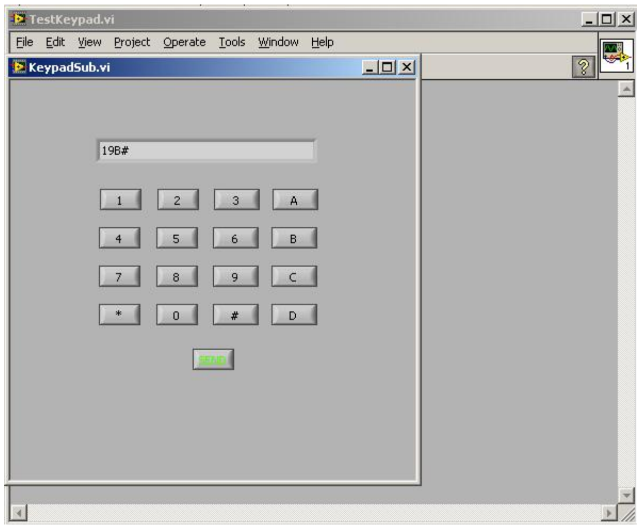


FIGURE 2
FRONT PANEL OF THE TESTKEYPAD.VI.

TestKeypad.vi, depicted in Figure (2) calls KeypadSub.vi, which will in turn monitor the key presses and display the key sequence until the Send button is pressed. When Send button is pressed, the computer generates and plays the unique tone combination of each key in sequence.

III. Digital Echo Processor

The echo of a signal can be implemented by a filter whose transfer function is

$$H(z) = 1 + az^{-D} \qquad (4).$$

where the parameter *a* represents the reflection and propagation losses such that $|a| \leq 1$ and the parameter *D* represents round-trip travel time from the source to the reflecting medium [15]. Figure (3) shows the front panel of Echo.vi, which implements a digital echo processor on one of the Windows system sound files.

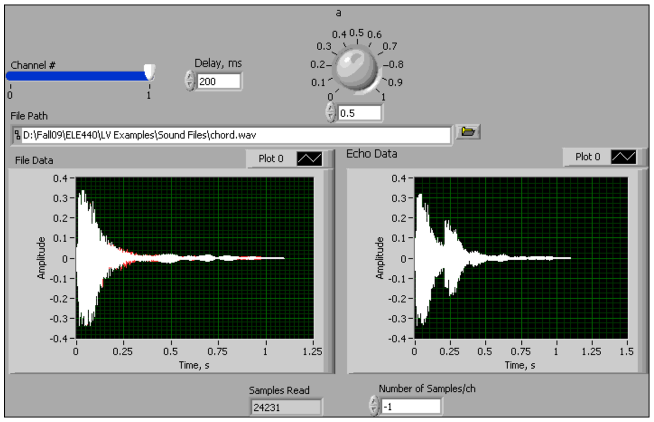

Channel # 1
Delay, ms 200
a 0.5
File Path D:\Fall09\ELE440\LV Examples\Sound Files\chord.wav
Samples Read 24231
Number of Samples/ch -1

FIGURE 3
FRONT PANEL OF ECHO.VI.

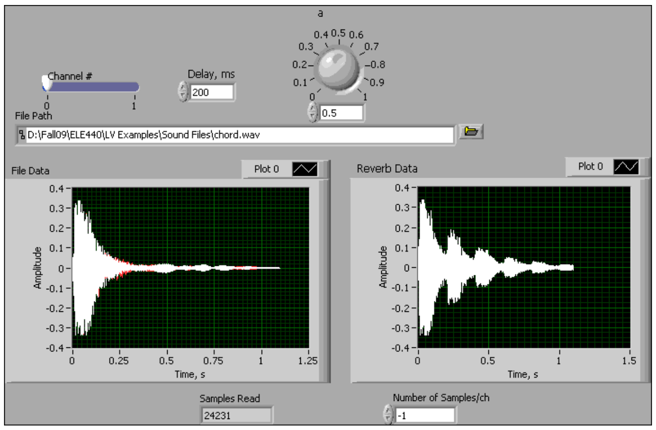

Channel # 0
Delay, ms 200
a 0.5
File Path D:\Fall09\ELE440\LV Examples\Sound Files\chord.wav
Samples Read 24231
Number of Samples/ch -1

FIGURE 4
REVERBERATION PATTERN FOR THE SAME SOUND FILE OF FIGURE (3).

IV. Plain Reverberator

Adding up an infinite number of successive echoes achieves the reverberating effect giving rise to an IIR comb filter whose transfer function is

$$H(z) = \frac{1}{1 - az^{-D}} \qquad (5)$$

where the parameters *a* and *D* are now attenuation and delay for each reverberation [15]. Figure (4) shows the reverberation pattern for the same sound file of Figure (3).

V. Flanging

In the days of reel tape players, flanging effect was created by playing the music simultaneously through two tape players and alternately pressing the flange of each tape reel to slow it down [15]. In the digital world, the flanging effect can be achieved by adding a delayed version of the signal to itself and allowing the delay, *D*, to vary in time:

$$y(n) = x(n) + ax(n - d(n)) \qquad (6)$$

where $y(n)$ is the flanged signal, $x(n)$ is the original signal and $d(n)$ is the time varying delay. One candidate for the delay is a sinusoidal variation within the limits $0 \leq d(n) \leq D$ of a low frequency F_d:

$$d(n) = \frac{D}{2}\left(1 - \cos(2\pi F_d n)\right) \qquad (7).$$

Figure (5) depicts the flanging effect on a pure sinusoidal signal.

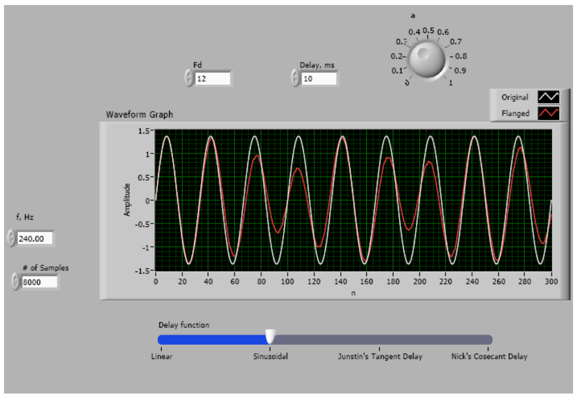

Fd 12
Delay, ms 10
f, Hz 240.00
# of Samples 8000
Delay function: Linear, Sinusoidal, Junstin's Tangent Delay, Nick's Cosecant Delay

FIGURE 5
FLANGING EFFECT ON A PURE SINUSOIDAL SIGNAL.

CONCLUSIONS

I have described my renewed interest in using LabVIEW as a teaching tool in a DSP course. I have given examples of the in-class activities I was able to bring out using the sound functions of LabVIEW 8.5. These are just a few of the many routines we implemented in class. However, when we started playing sound files, I was able to observe the renewed interest in the students' participation. We had covered many algorithms and graphed countless waveforms leading up to these examples with sound, but mere graphs were not able to capture students' attention as much as the audio effects. Since I had very few students in this class, I was able to administer the final exam orally. I deliberately asked each student what they had learned in this class. It is interesting that the students' recollections from earlier chapters had several gaps, but they all recounted the subjects we covered in the digital audio effects chapter with consistent accuracy. I plan to introduce more audio examples in earlier chapters in my future offerings of this course.

REFERENCES

[1] Nilsson, J. W. and Riedel, S. A. *Electric Circuits*, *Ninth Edition*, Upper Saddle River, NJ: Prentice Hall (2009)

[2] Rashid, M. H., *Spice for Power Electronics and Electric Power*, Engle wood Cliffs, NJ: Prentice Hall (1993).

[3] Sedra, A. S. and Smith, K. C., *Microelectronic Circuits*, *Fourth Edition*, New York, NY: Oxford University Press (1998)

[4] Roberts, G. W. and Sedra, A. S., *Spice for Microelectronic Circuits Third Edition by Sedra/Smith*, New York, NY: Oxford University Press (1992).

[5] Howe, R. T.and Sodini, C. G., *Microelectronics – An Integrated Approach*, Upper Saddle River, NJ: Prentice Hall (1997).

[6] Franco, S., *Electric Circuit Fundamentals*, Philadelphia, PA: Saunders College Publishing (1995).

[7] Jaeger, R. C., *Microelectronic Circuit Design*, New York, NY: McGraw-Hill (1997).

[8] Nise, N. S., *Control Systems Engineering*, New York, NY: John Wiley & Sons (2000).

[9] Ogata, K, *Modern Control Engineering*, Upper Saddle River, NJ: Prentice Hall (1997).

[10] Dorf, R. C. and Bishop, R.H., *Modern Control Engineering, 8th Ed.*, Reading, MA: Addison Wesley (1998).

[11] Palm III, W. J., *Modeling, Analysis, and Control of Dynamic Systems, 2nd Edition*, New York, NY: John Wiley & Sons (2000).

[12] Couch II, L. W., *Digital and Analog Communication Systems, Sixth Edition*, Upper Saddle River, NJ: Prentice Hall (2001).

[13] Bateman, A., *Digital Communications: Design for the Real World, 1/e*, Upper Saddle River, NJ: Prentice Hall (1999).

[14] Anderson, J. B., *Digital Transmission Engineering, 1/e*, Upper Saddle River, NJ: Prentice Hall (1999).

[15] Orfanidis, S. J., *Introduction to Signal Processing*, Upper Saddle River, NJ: Prentice Hall (1996).

[16] Moon, T. K. and Stirling, W. C., *Mathematical Methods and Algorithms for Signal Processing*, Upper Saddle River, NJ: Prentice Hall (2000).

[17] Tanyel, M., *Engineering Explorations with LabVIEW*, Philadelphia, PA: Harcourt Brace Custom Publishers (1994).

[18] Tanyel, M., "Virtual Experimentation in Freshman and Sophomore Years," in *Proceedings of 58th Annual ASEE North Midwest Section Meeting*, Oct. 1996.

[19] Abu Zeid, O. A., Tanyel, M., "Innovation in Teaching Mechanical Engineering Applications", in *Proceedings of 1994 Frontiers in Education Conference*, pp. 82-86, Nov. 1994.

[20] Scoles, K., Tanyel, M., Onaral, B., "Computing in Electrical Engineering Education at Drexel University", in IEEE Transactions on Education, vol. 36, no. 1, pp. 198-203, Feb. 1993.

[21] Tanyel, M., Quinn, R., Barge, E., "An Engineering Laboratory for Freshmen - Computer Utilization", in *1990 ASEE Annual Conference Proceedings*, Toronto, June 26-29 1990.

[22] Chugani, M. L., Samant, A. R., Cerna, M., *LabVIEW Signal Processing*, Upper Saddle River, NJ: Prentice Hall (1998).

AUTHOR INFORMATION

Murat Tanyel, Professor of Engineering, Geneva College, Beaver Falls, PA, 15010, mtanyel@geneva.edu.