FROM BLOCK DIAGRAMS TO GRAPHICAL PROGRAMS IN DSP

Marlin Viss, Murat Tanyel
Dordt College

Abstract

Recent Control Systems, Communication Systems and Digital Signal Processing (DSP) courses have relied heavily on MATLAB and/or C, representing the state of the art in textual programming, for their standard computer tools. Many textbooks are published containing examples, if not sections, utilizing these textual languages. Whereas this environment may be efficient in manipulating equations, textual implementation of processes best described by block diagrams loses its intuitive substance. In this paper, we will describe experiences in a DSP course with an alternative graphical programming environment, namely LabVIEW, from both a student's and an instructor's perspective. We will describe the adjustments that have to be made by individuals trained in conventional, textual programming environments during the transition to the graphical environment. We will give examples of implementations that are better left graphical, such as direct form, canonical, transpose of canonical and cascade realizations of IIR filters. We will conclude with a summary of student feedback on the effectiveness of the graphical programming environment in the presentation of DSP topics.

I. Introduction

As computer applications become indispensable tools in electrical engineering curriculum, we observe that a number of applications have become widespread computer tools in electrical engineering textbooks. Spice and its derivatives pervade courses that cover circuit analysis and electronics, with most standard textbooks on these subjects devoting sections or having supplements available with simulations in this application[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] and even circuit analysis[1]. The C programming language has replaced FORTRAN in the electrical engineering curriculum, as the more senior author has observed this transition from his undergraduate studies in the late seventies to graduate studies in the eighties. Numerical recipes in C, either in software or printed book form[17], have helped many a graduate student in getting through different projects. With the exception of SIMULINK and the graphical interface for PSpice, 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 Hewlett-Packard's HP VEE, available on Windows, HP-UX and SunOS[18]. Another serious contender is SIMULINK with its textual roots on MATLAB.

All these graphical environments are data-driven as opposed to the command-driven textual languages, in which the computer executes one line of command after another. Therefore, processes that describe what happens to various inputs to achieve an output, so easily 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. On the other hand, processes that are best described by line after line of mathematical equations are better suited for textual programming environments. That is not to say that block diagram representations cannot be realized by textual languages and vice versa. DSP applications traditionally implemented in C or MATLAB are testimony to the effectiveness of the former, but the intuitive association of the process to the code that realizes/simulates it becomes more obscure when the code is in a textual language. Likewise, processes that involve many complicated mathematical formulae become cumbersome to code in graphical languages.

This paper describes the recent experience we have had in using LabVIEW, a graphical programming environment, in a senior level DSP course at Dordt College. We will first give a brief course description with the particular textbook used. We will then give an overview of DSP utilities of LabVIEW. We will proceed with examples in which a graphical environment is more intuitive for the student of DSP, who has to be able to program signal processing routines on top of knowing how to use them. We will then convey the consensus of our small class on the effectiveness of LabVIEW in demonstrating DSP concepts, finishing with our concluding remarks.

II. The Overall Environment: The Course, the Facilities and the Tools

The engineering department at Dordt College offers an ABET-accredited engineering major as well as a non-accredited engineering science major. Students in the engineering major may opt for either a mechanical or an electrical emphasis. As a result of the curriculum revision that has recently been implemented, students in the electrical emphasis have a number of class-electives in their senior year. In the first semester the class chooses between Analog Circuit Design and Digital Signal Processing. This year's senior class was the first to have this choice and they decided to go digital, inaugurating the full-semester offering of DSP. The textbook that the instructor chose for this first offering was Orfanidis' *Introduction to Signal Processing*[15]. This textbook was chosen because of 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.

EGR 366, Digital Signal Processing, is a three credit class which met for three 50-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. After the initial background material was covered in a conventional classroom setting, the second (and larger) portion of the semester was spent in the electronics laboratory, which is furnished with computers with the LabVIEW software.

The Dordt College Engineering Department enjoys two facilities dedicated to engineering applications. One is a computer lab that offers, on top of the general college-wide applications such as word processing and internet access, engineering specific applications such as AutoCAD, MathCAD (v. 8) and MATLAB (v. 6) that are served over the network. The other facility is the electronics lab which is comprised of 11 workstations furnished with computers that are connected to the Dordt network and therefore have access to network-served applications plus the National Instruments' full development version 5.1 of LabVIEW to work with the NI-488.2 (IEEE 488 standard) cards for the control of and communication with the Tektronix TDS 210 Digital Oscilloscopes and Global 2003 Sythesized Function Generators. During the first few weeks of the semester the class agreed to meet in the electronics lab in addition to regularly scheduled times to learn LabVIEW with the intent of using it for applications in DSP. LabVIEW was chosen to supplement the course for two main reasons: i) the professor's prior experience with it[19-23] and ii) the students' recognition that it is a worthwhile programming environment that will complement the other packages, namely Visual Basic, MATLAB and MathCAD, 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[24]. The three important components of such applications are data acquisition, data analysis and data visualization. LabVIEW offers an environment which 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. 5.1) contains VIs that are arranged in groups. The groups pertinent to the DSP course are listed below:

- **Signal Generation**, containing VIs for generating different signals such as a sine wave, a square wave, a chirp signal and white noise.

- **Time Domain** [processing], containing VIs for computing convolution, deconvolution, auto-correlation, cross-correlation, decimation and similar routines.

- **Frequency Domain** [processing], containing VIs for computing the power spectrum, complex FFT, complex inverse FFT, fast Hilbert transform, inverse fast Hilbert transform and similar routines.

- **Measurement** containing such procedures as auto power spectrum, cross power spectrum, amplitude and phase spectrum, [estimation of] transfer function, [estimation of] impulse function.

- **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.)

- **Windows** containing such windows as Hanning, Hamming, Triangle, Blackman, Kaiser, etc.

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.

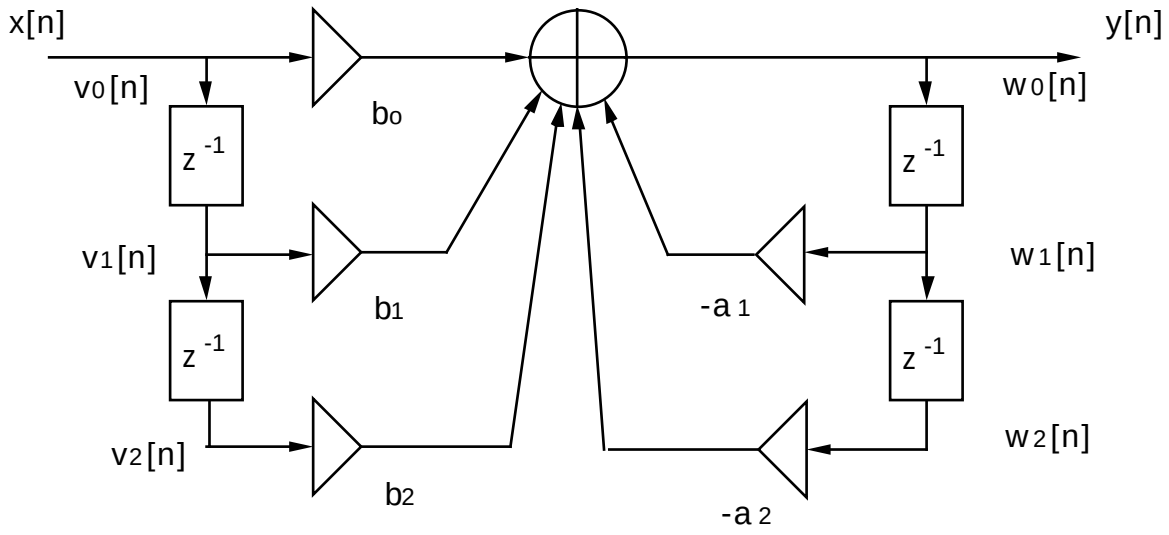


Figure 1: Direct form realization of a second-order IIR filter, as depicted in a standard textbook.

III. Examples Better Suited for Graphical Programs

One subject that is covered in this course is the different realizations of IIR filters, such as direct form and canonical form realizations. Although LabVIEW has VIs that will compute coefficients for cascade realizations of some typical filters and another VI that will filter an incoming data stream using these coefficients, the whole structure of the filter is transparent to the user. Therefore we decided to write our own programs that would implement such realizations, thus having more fundamental VIs in our toolkit.

A simple second order filter has the transfer function[15]

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \qquad (1).$$

This transfer function in the *z*-domain would result in an I/O difference equation of

$$y[n] = -a_1 y[n-1] - a_2 y[n-2] + b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] \qquad (2).$$

The direct form realization for this filter is depicted, in block diagram form, in Fig. 1. This second order filter can be implemented in pseudo-code by the following sample processing algorithm[15]:

```
for each input sample x[n] do:
   v[0] = x[n]
   w[0] = -a[1]w[1] - a[2]w[2] + b[0]v[0] + b[1]v[1] + b[2]v[2]|
   y[n] = w[0]
   v[2] = v[1]
   w[2] = w[1]
   v[1] = v[0]
   w[1] = w[0]
end
```

The LabVIEW realization of the same algorithm is depicted in Fig. 2. This realization employs a *for loop*, which gets executed as many times as the number of elements of the array *x* coming to it (in LabVIEW terminology, *indexing* is *enabled*). The loop employs two distinct *shift registers* (LabVIEW term for temporary storage for values), one for the array *w* and one for *v*. To a programmer familiar with LabVIEW the correspondence between Fig. 1 and Fig. 2 is stronger than the correspondence of Fig. 1 to the textual pseudo code given above.

Fig. 3 depicts the block diagram for the canonical realization for a second order filter. The canonical form realization can be implemented with the pseudo code as follows:

```
for each input sample x[n] do:
   w[0] = x[n] -a[1]w[1] - a[2]w[2]
   y[n] = b[0]w[0] + b[1]w[1] + b[2]w[2]
   w[2] = w[1]
   w[1] = w[0]
end
```

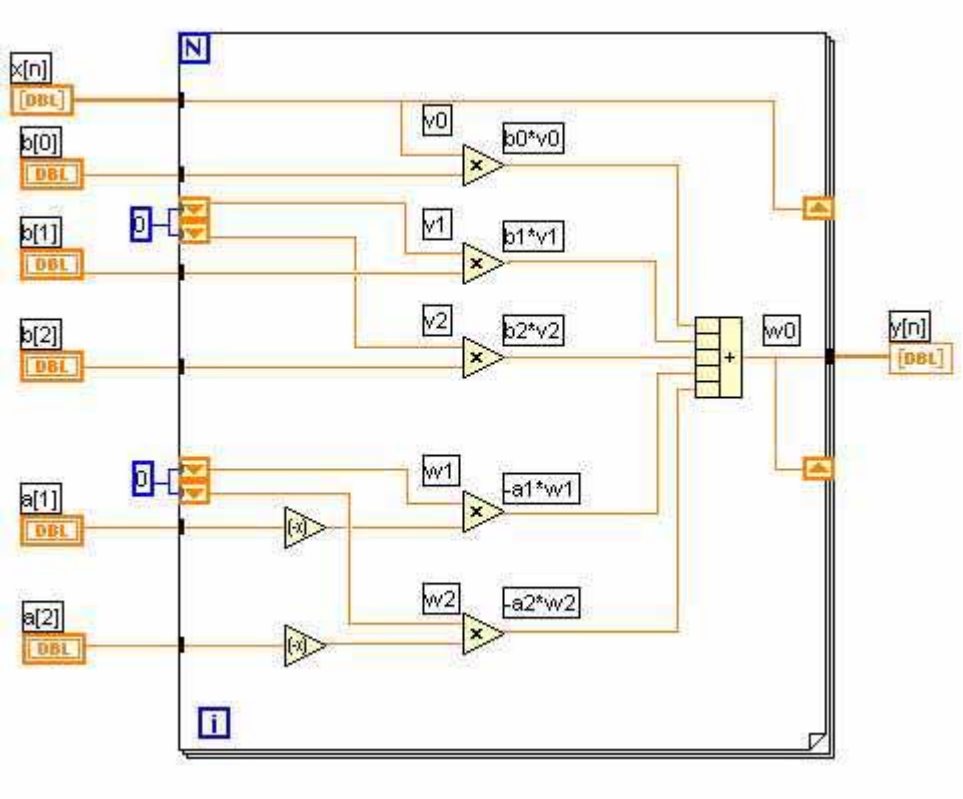


Figure 2: Direct form realization of a second order IIR filter in LabVIEW.

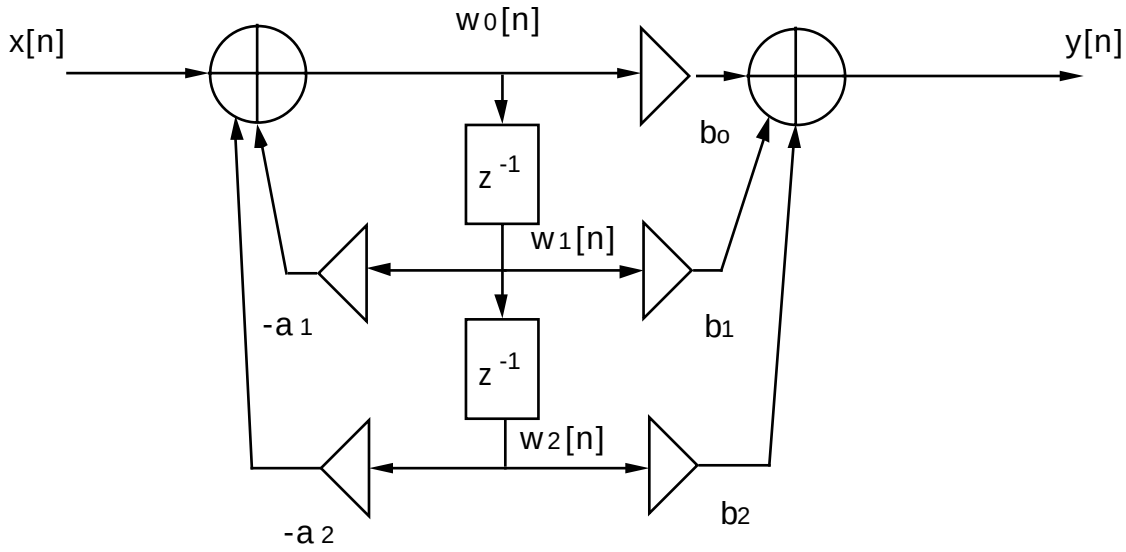


Figure 3: Canonical form realization of a second-order IIR filter.

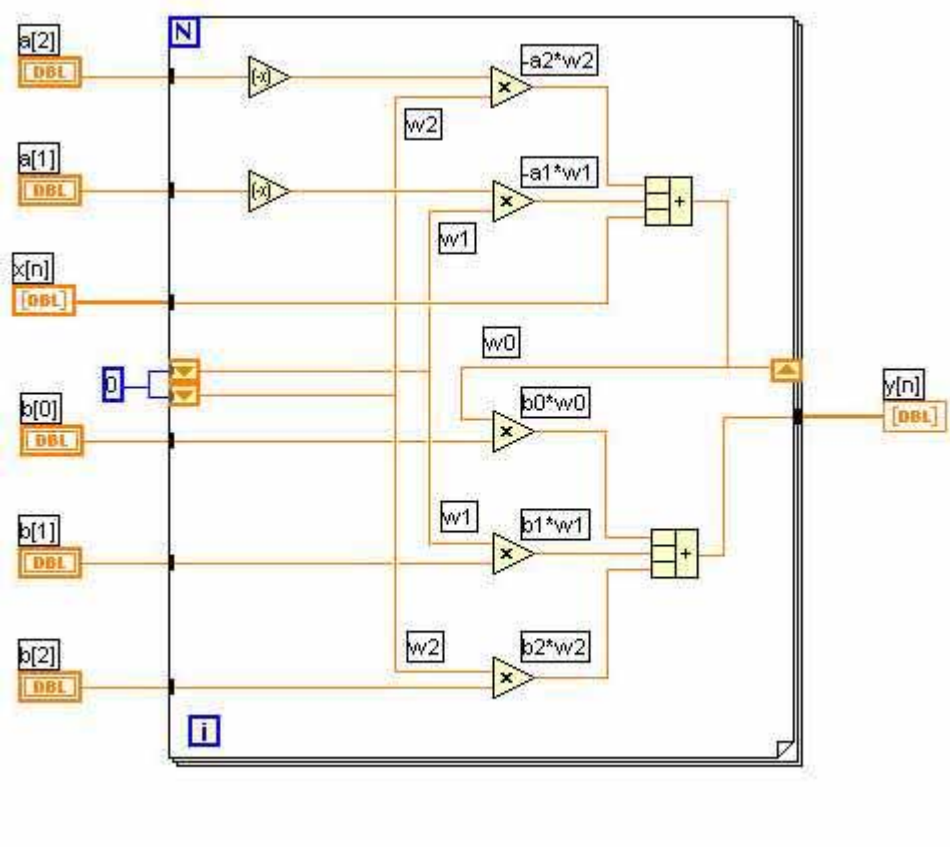


Figure 4: Canonical form realization for a second order IIR filter in LabVIEW.

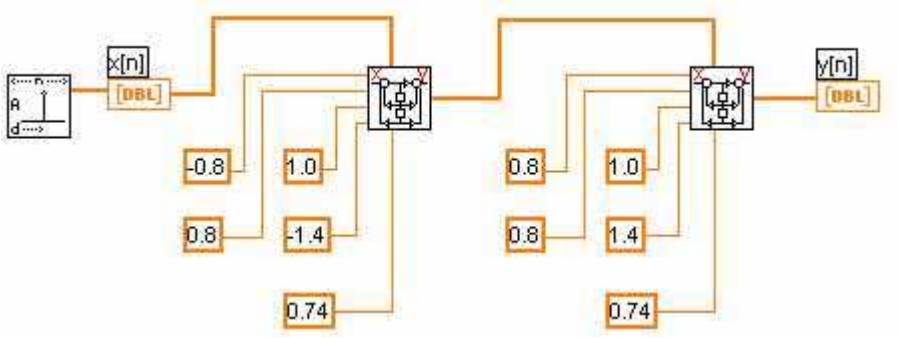
Fig. 4 depicts the LabVIEW program that realizes the filter in Fig. 3. Note that there are now two summations but only one shift register, corresponding directly to the elements in the block diagram.

In general, a filter of arbitrary order can be realized through a cascade of second order sections. An example in[15] (Example 7.4.2, p. 290) takes a fourth order filter defined by the transfer function

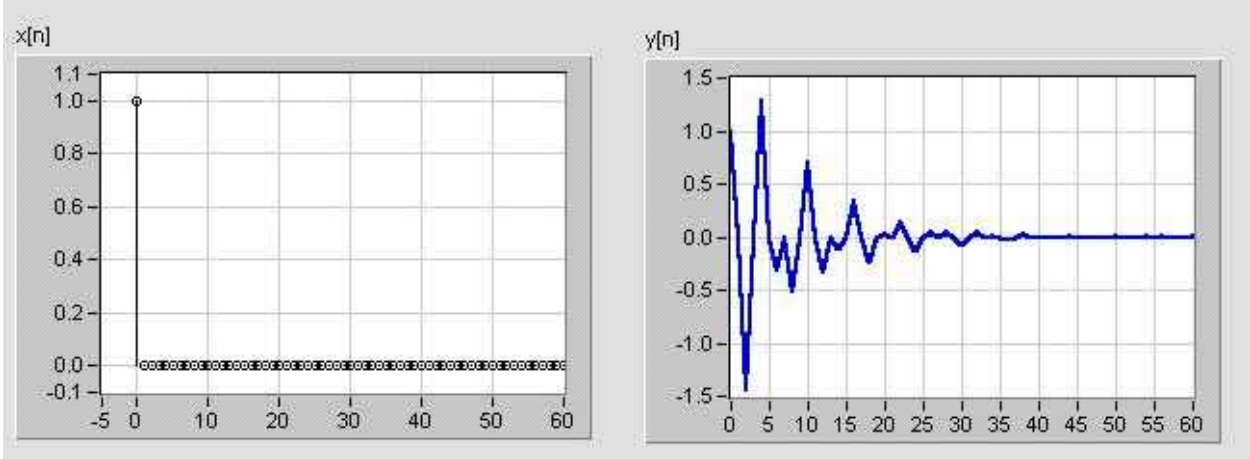
$$H(z) = \frac{1 - 0.48z^{-2} + 0.5476z^{-4}}{1 + 0.96z^{-2} + 0.64z^{-4}} \qquad (3)$$

and determines the cascade form of the filter as

$$H(z) = \frac{1 - 1.4z^{-1} + 0.74z^{-2}}{1 - 0.8z^{-1} + 0.8z^{-2}} \cdot \frac{1 + 1.4z^{-1} + 0.74z^{-2}}{1 + 0.8z^{-1} + 0.8z^{-2}} \qquad (4).$$



(a)



(b)

Figure 5: a) Cascade realization of a filter with second order canonical sections, b) The front panel showing the input (an impulse) and the output (the impulse response).

Fig. 5-a depicts the LabVIEW realization of this fourth order filter by cascading two second order canonical form sections as we try to obtain its impulse response. The inputs to the first section are the output of the impulse generator (the signal) and the *a* and *b* coefficients of the first component of equation (4). The output of the first section is the signal input to the other section with the appropriate coefficients as *a* and *b* inputs. The output of the second section is the final output and goes to a plotting block (*Waveform Graph* in LabVIEW). Note that the graphical program is so self apparent that it can almost be used as a general block-diagram representation for this example. Fig. 5-b shows the front panel of this VI with the input (the impulse) and the output (impulse response) of the filter. The ease with which graphs can be obtained, and the ease with which they can be formatted to look like the output of a realistic instrument generates excitement in students.

IV. The Impressions of the Students

Apart from spontaneous comments of the students, a survey was conducted to gauge their response to the presentation of DSP topics in LabVIEW. The verbal comments uttered inside the laboratory and in the corridors indicated that in general, the students welcomed the idea and were even having fun doing it! Because of the small size of the class, the survey results would not be statistically insignificant and are treated as documented anecdotes for this paper. According to the survey results, all of the students were familiar with a textual programming language prior to this class. The self-reported skill level was, on the average, "generating moderately complex programs". As a result of exposure in this course, all students indicated that they were able to write simple programs in LabVIEW. When asked for what tasks textual languages are better suited than LabVIEW, all agreed that mathematical formulae are better handled by textual languages. One indicated that even a simple formula might take up a lot of space on the screen when implemented in LabVIEW and that even the use of a formula node requires many "connections" to the node, complicating the program.

All of the students appreciated the visual feedback LabVIEW provides. They all indicated the ease of "block diagram programming" and noted the abundance of DSP algorithms provided and their ease of use. They all indicated that it is very easy to see the connection of a LabVIEW program to a block diagram. The question on the connection of a textual program to the block diagram got mixed responses. One indicated that once a block processing algorithm is found, the connection is easy but failed to comment how typical block processing algorithms correspond to the process. One indicated that the language the textbook used, namely C, was enigmatic to him. In general, the students did not find the connection as intuitive as LabVIEW.

V. Conclusions

The results of the survey did not reveal any information that the authors did not expect. Indeed, mathematical formulae are better dealt with by textual languages and there are some algorithms, such as the implementation of circular buffers, that are very simple in the C language but present great difficulty in LabVIEW. Therefore, we should not abandon the use of such languages, but use every tool in applications where its strengths excel. However, the instrument-like user interface of LabVIEW, its abundance of analysis VIs make it very attractive and inviting to implement most DSP routines. In applications where it is cumbersome, one of two routes can be chosen: write the program in C and import it into LabVIEW through its *Command Interface Node*, which can be done by someone experienced in both languages, or have the experienced programmer write the painstakingly difficult algorithm in LabVIEW and then make it available as a subVI to the rest of the class so that students do not get bogged down by a few non-intuitive programs. In either case, this activity would add to the richness of DSP algorithms available for LabVIEW and will no doubt make the utilization of the graphical interface in DSP, an already pleasant experience, even more pleasant.

ACKNOWLEGEMENTS: The LabVIEW software used in this course, as well as the NI-488.2 cards and Tektronix TDS 210 Digital Oscilloscopes in the electronics lab were purchased through a grant from Johnson Controls in Holland, MI. The authors would like to thank Johnson Controls for the upgrade of the electronics lab. The authors would also like to thank the three other students in the DSP class, Eric Eekhoff, James Grossman and Seth VerMulm for sharing their impressions frankly.

Bibliography

1. Nilsson, J. W. and Riedel, S. A. *Electric Circuits, Sixth Edition*, Upper Saddle River, NJ: Prentice Hall (2000).

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. Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P., *Numerical Recipes in C : The Art of Scientific Computing*, Cambridge: Cambridge University Press (1993)

18. Helsel, R., *Cutting Your Test Development Time with HP VEE*, Englewood Cliffs: Prentice Hall (1994).

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

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

21. 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.

22. 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.

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

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

MARLIN VISS
Marlin Viss is a senior engineering student at Dordt College. He is expected to obtain his B. S. in engineering with an electrical emphasis in May 2001. He spent several years programming in C and PERL for a consulting firm, and works as a TA for the engineering programming courses at Dordt College. He plans to continue DSP studies in graduate school.

MURAT TANYEL
Murat Tanyel is a professor of engineering at Dordt College. He teaches upper level electrical engineering courses. Prior to teaching at Dordt College, Dr. Tanyel taught at Drexel University where he worked for the *Enhanced Educational Experience for Engineering Students* (E^4) project, setting up and teaching laboratory and hands-on computer experiments for engineering freshmen and sophomores. For one semester, he was also a visiting professor at the United Arab Emirates University in Al-Ain, UAE where he helped set up an innovative introductory engineering curriculum. Dr. Tanyel received his B. S. degree in electrical engineering from Boğaziçi University, Istanbul, Turkey in 1981, his M. S. degree in electrical engineering from Bucknell University, Lewisburg, PA in 1985 and his Ph. D. in biomedical engineering from Drexel University, Philadelphia, PA in 1990.

DSP Builder consists of several Simulink* libraries that allow you to implement DSP designs quickly and easily. DSP Builder implements the hardware as VHDL or Verilog HDL with scripts that integrate with the software and the simulator.Â 2. DSP Builder for Intel FPGAs Advanced Blockset Getting Started. 2.1. Starting DSP Builder in MATLAB. 2.2. Browsing DSP Builder Libraries and Adding Blocks to a New Model. 2.3. Browsing and Opening DSP Builder Design Examples. 2.4. Creating a New DSP Builder Design with the DSP Builder New Model Wizard. Recent Control Systems, Communication Systems and Digital Signal Processing (DSP) courses have relied heavily on MATLAB and/or C, representing the state of the art in textual programming, for their standard computer tools. Many textbooks are published containing examples, if not sections, utilizing these textual languages. Whereas this environment may be efficient in manipulating equations, textual implementation of processes best described by block diagrams loses its intuitive substance.Â We will conclude with a summary of student feedback on the effectiveness of the graphical programming environment in the presentation of DSP topics. Discover the world's research. 17+ million members. Graphical programming for DSPs Graphical programming can help you reduce time to market, improve code reuse and port your design to multiple platforms. Hereâ€™s how.Â DSP-based system-on-chip moves speech recognition from the lab to portable devices Researchers from Tsinghua and Infineon Technologies describe development of a speech recognition system-on-chip for use in consumer applications such as toys, voice-based remote control and speech recorders. Why Multiprocessor DSP Systems Need CORBA CORBA enables software components in a multiprocessor system to easily communicateâ€”regardless of what language they are written in, what OS they run on, or where they are located.