AC 2007-1705: A SINGLE PLATFORM TO TEACH CIRCUIT DESIGN, BIOINSTRUMENTATION, CONTROL & SIGNAL PROCESSING IN BIOMEDICAL ENGINEERING

Shekhar Sharad, National Instruments

A Single Platform to Teach Circuit Design, Bioinstrumentation, Control & Signal Processing in Biomedical Engineering

Traditional Biomedical Engineering programs use multiple software platforms to teach biomedical engineering concepts in circuit design, bioinstrumentation, control and signal processing. As a result, the students spend a lot of time learning the different tools instead of learning the concepts. With the evolution of graphical programming tools, it is now possible to provide a single graphical platform that spans all facets of engineering such as circuit design, data acquisition, instrument control, control systems, digital signal processing and senior design. This results in the students spending much less time learning different tools and more time learning and implementing the concepts in a hands-on environment. In this paper, we will outline and explain the different components and advantages of such an Integrated Biomedical Engineering Platform. We will also show example implementation of such an integrated graphical platform in areas such as Bioinstrumentation and Circuit Design

1. Introduction

In the rapidly changing field of Biomedical Engineering (BME), there is a continual challenge to teach concepts from an ever-increasing set of courses that span curricula from multiple disciplines. For example, a typical BME program[1,2,3] today overlaps with the courses from electrical engineering, mechanical engineering and the sciences. However, the time period to cover all the concepts is still the same as before which presents the challenge to find ways to be efficient with the time spent learning tools versus learning concepts. With graphical programming tools evolving rapidly in the past few years, it is now possible to have a single platform that can span multiple courses and senior design projects. In this paper, we will illustrate the use of graphical programming platforms to teach concepts across multiple courses, with an example from bioinstrumentation and circuit design.

2. Tool-fatigue

One of the most significant challenges that educators face today in teaching different courses from different domains is the use of multiple tools. For example, students may use one tool for design and simulation of their algorithm, while they may need to rewrite their application to suit another tool for deployment in a design project. For example, most universities ask students to use one tool for data acquisition(LabVIEW) and another for the signal processing(Matlab®) and yet another tool to deploy to hardware(Xilinx tools). The challenge with this approach is that students end up spending the majority of their time learning new tools and less time on actually learning the concepts. This leads to what can be referred to as "Tool-fatigue", where the student knows many tools but not enough domain knowledge to know how to solve the problems with the right approach. Tool-fatigue affects biomedical engineering more than other disciplines because of the overlap between other disciplines – A BME student needs to learn not only tools from the Electrical engineering domain, but also from mechanical engineering and from the sciences.

In addition, BME students generally do not have extensive experience with textual programming languages such as C and C++. Hence, the degree of learning is higher when it comes to learning text-based languages that require extensive programming expertise. This is where innovative approaches such as graphical programming have significant advantages in helping teach concepts to BME students.

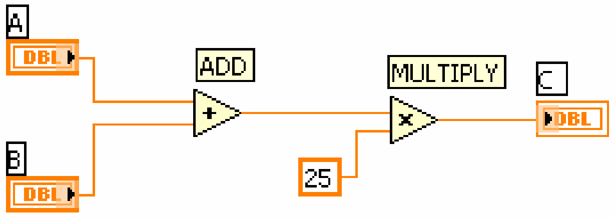
3. Graphical Programming: An Integrated Approach

One of the natural mechanisms that engineering students use to learn concepts is block diagrams. In the programming world, the paradigm that they use is generally referred to as dataflow paradigm[4,5]. Under the dataflow paradigm, the solution to a problem is modeled as a series of operations (represented as blocks) with the data seemingly invisible. While this may seem a minor detail, it helps provide a much higher level of abstraction when designing algorithms and systems.

Figure 1 shows the difference between a C-code snippet to add two numbers and multiple the result by 25 and the dataflow representation of the same.

```
int func_add_multiply(a,b)
{
//program to add and multiple
int c=0;
c=a+b;
c=c*25;
return c;
}
```

(a) Textual Representation

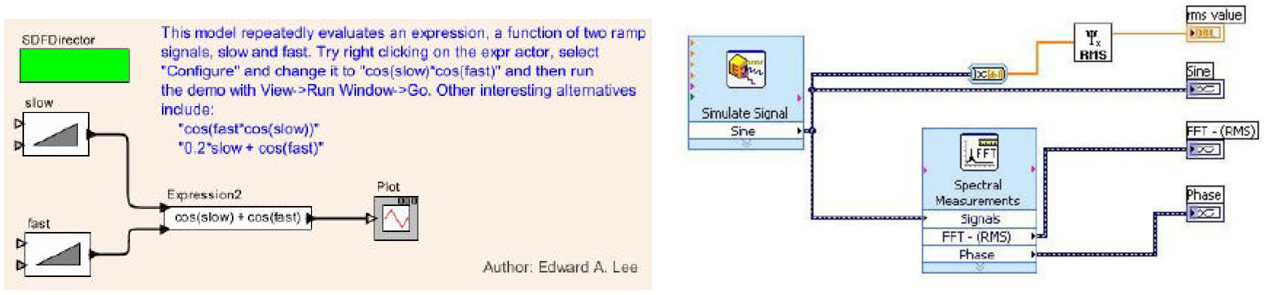


(b)Equivalent Dataflow Representation

Figure 1. Comparison Between Textual and Dataflow Paradigms

Several key points are worth noting in the two representations. First, textual representation needs a lot of attention to syntax and variables. In the example in figure 1, the variable c needs to be properly declared and identified (int c=0). Secondly, the textual approach uses a procedural approach with one function following the other with no apparent "flow" of data making it difficult to comprehend the objective of a program, whereas the dataflow representation gives everyone a clear overview of what is happening. When professors teach concepts to students, they use a block diagram approach as shown in figure 1(b). Graphical programming helps take advantage of this natural mechanism to not only teach concepts but also to simulate and prototype applications in BME.

Graphical programming, based on the dataflow paradigm, provides a simple yet powerful and interactive environment that educators and students can use to teach and learn concepts. Graphical programming languages have also evolved to be able to interface with multiple hardware platforms and instruments such as FPGAs, DSPs, data acquisition systems, traditional benchtop instruments and medical instrumentation. This flexibility in connecting to hardware and the availability of user-defined and pre-built functions for different areas from circuit design to signal processing makes graphical programming a key methodology to enable students to focus more on concepts and less on learning tools. Figure 2 shows some of the leading graphical programming languages.



This model repeatedly evaluates an expression, a function of two ramp signals, slow and fast. Try right clicking on the expr actor, select "Configure" and change it to "cos(slow)*cos(fast)" and then run the demo with View->Run Window->Go. Other interesting alternatives include:
"cos(fast*cos(slow))"
"0.2*slow + cos(fast)"

Figure 2. Examples of Graphical Programming Languages (a) Ptolemy[6] (b) NI LabVIEW[7]

Figure 3(a) shows the comparison between a typical BME curriculum[1,2,3] that would include traditional tools for some courses and how graphical programming provides a single, integrated software platform

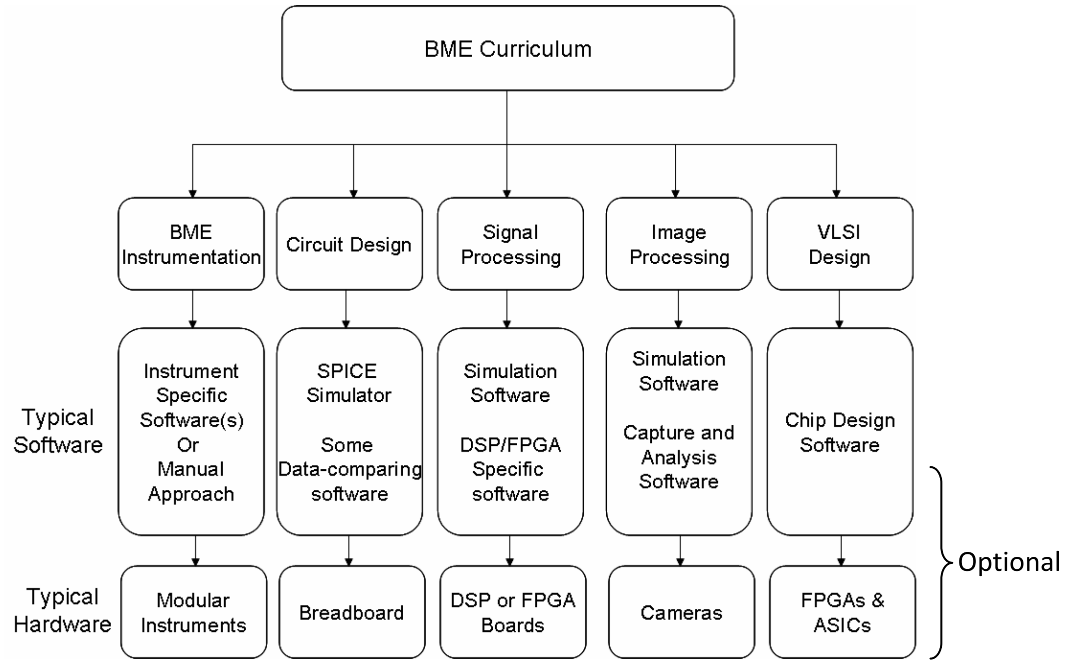


Figure 3(a) Courses from a Typical BME Curriculum with Unique Software & Hardware Needs

Figure 3(b) brings out some important benefits that graphical programming offers educators in BME. Graphical programming uses a uniform syntax for all areas which means that a student does not have to learn a new programming language. In addition, graphical programming languages such as NI LabVIEW connect to a wide range of hardware from medical instruments to FPGA-based platforms and cameras for image processing. Let us now look at some of these implementations in detail

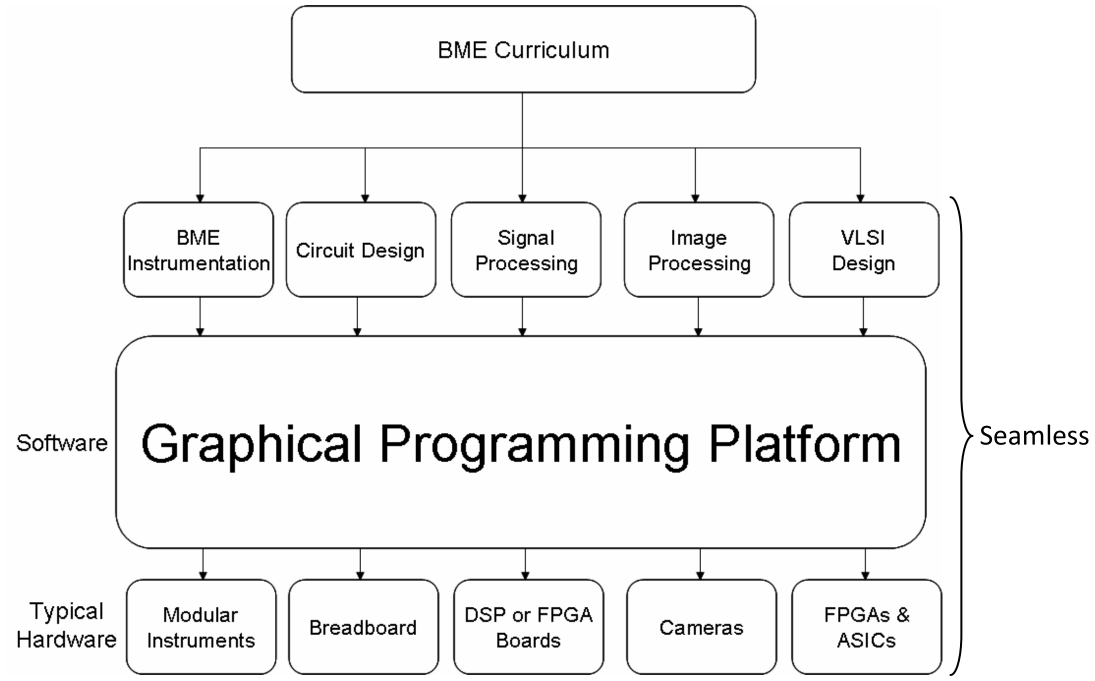


Figure 3(b) The Courses in Figure 2(a) with Graphical Programming

4. Graphical Programming for Bioinstrumentation and Circuit Design

The Bioinstrumentation curriculum partly deals with acquiring and analyzing data from various sensors and instruments and forms one of the core courses that every BME student needs to go through. Bioinstrumentation poses an interesting challenge to educators with the vast number of sensors and instruments that one can connect to. Figure 4a illustrates the traditional approach where data from multiple sensors needs to be acquired and analyzed.

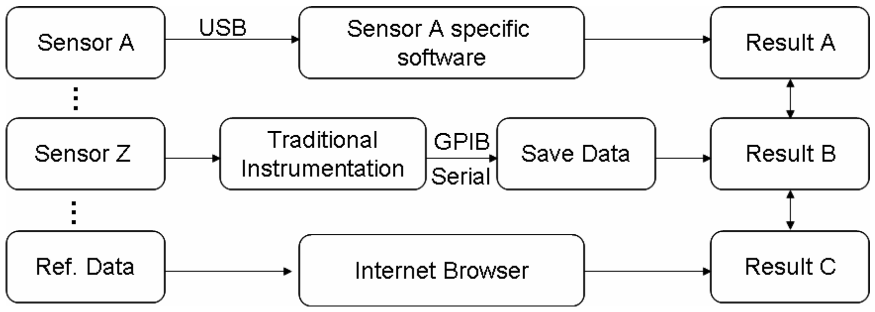


Figure 4a. Traditional Approach to Acquire Data From Multiple Sensors

In the traditional approach, there would be multiple paths to acquiring data from each sensor, such as using sensor specific software and acquiring over USB or using GPIB or Serial buses to communicate with traditional instruments to import data into a computer program. Afterwards, data from multiple sources needs to be compiled to be compared.

Figure 4b shows the same course that uses a graphical programming platform.

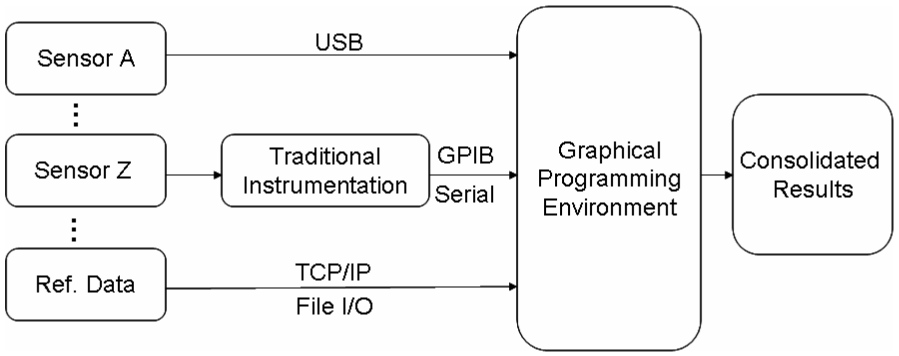


Figure 4b. Effect of Graphical Programming Environment on Bioinstrumentation

Graphical programming environments include capabilities to interface with multiple buses included USB, Serial and GPIB. Furthermore, it is possible to define a custom communication pattern for a specific sensor via serial or GPIB. In addition, graphical programming environments include some kind of plug-ins to connect to the web via TCP/IP. This enables students to be able to acquire data from a variety of sources, consolidate and compare them automatically under a single platform. Figure 5 shows an example of a graphical program in NI LabVIEW that acquires data from two sensors, one connected to a data acquisition device and another to a traditional instrument that communicates via GPIB.

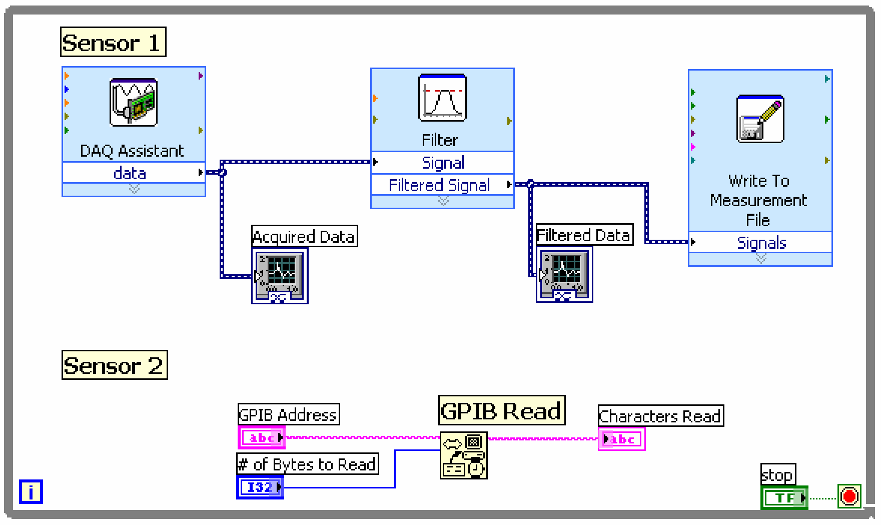


Figure 5. Acquiring Data from Two Sensors Via GPIB and PCI Buses

Another engineering area that is closely related to bioinstrumentation is circuit design. Traditional tools for circuit design do not offer a seamless environment to compare simulation data with real-world signals. Students would have to simulate the system in a SPICE simulation environment such as NI Multisim, save the data and then retrieve it in an analysis package. With graphical programming, educators now have access to a seamless platform from design, to prototyping and comparison of results between the simulation domain and the real world. Figure

6 shows the seamless integration from design to prototyping for circuit design. The SPICE simulation tool used here is NI Multisim, but any other simulation package can be used as well..

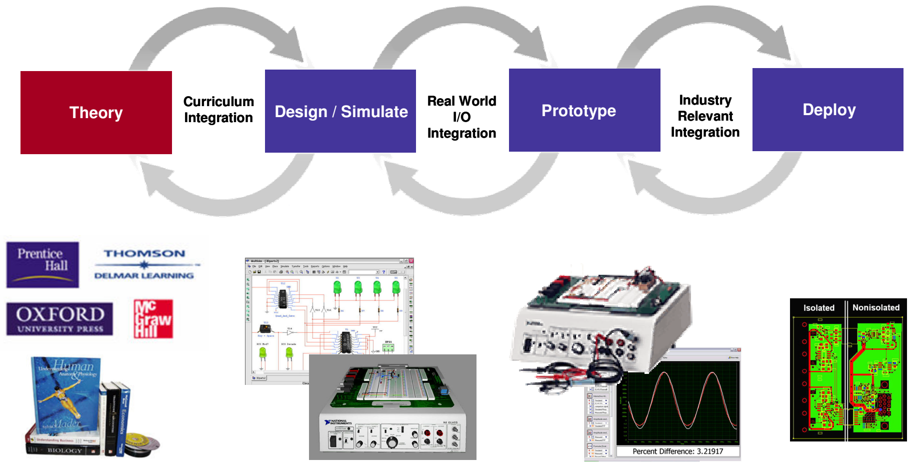


Figure 6. Seamless Integration Between Theory, Design and Prototyping

To illustrate, one of the basic concepts that every student learns is analysis of RLC circuits. The power of the seamless integration with graphical programming is demonstrated when this example is applied to it – the students first learn the concepts from leading textbooks. Then they use SPICE simulation and capture software such as NI Multisim to design and simulate the RLC circuit and understand the step response. In order to verify the simulation results, they then prototype the RLC circuit on a prototyping platform such as NI ELVIS and compare the simulation results with real-world data using a graphical programming environment such as NI LabVIEW. Hence, the students can now bring theory to life and understand the circuit design concepts better.

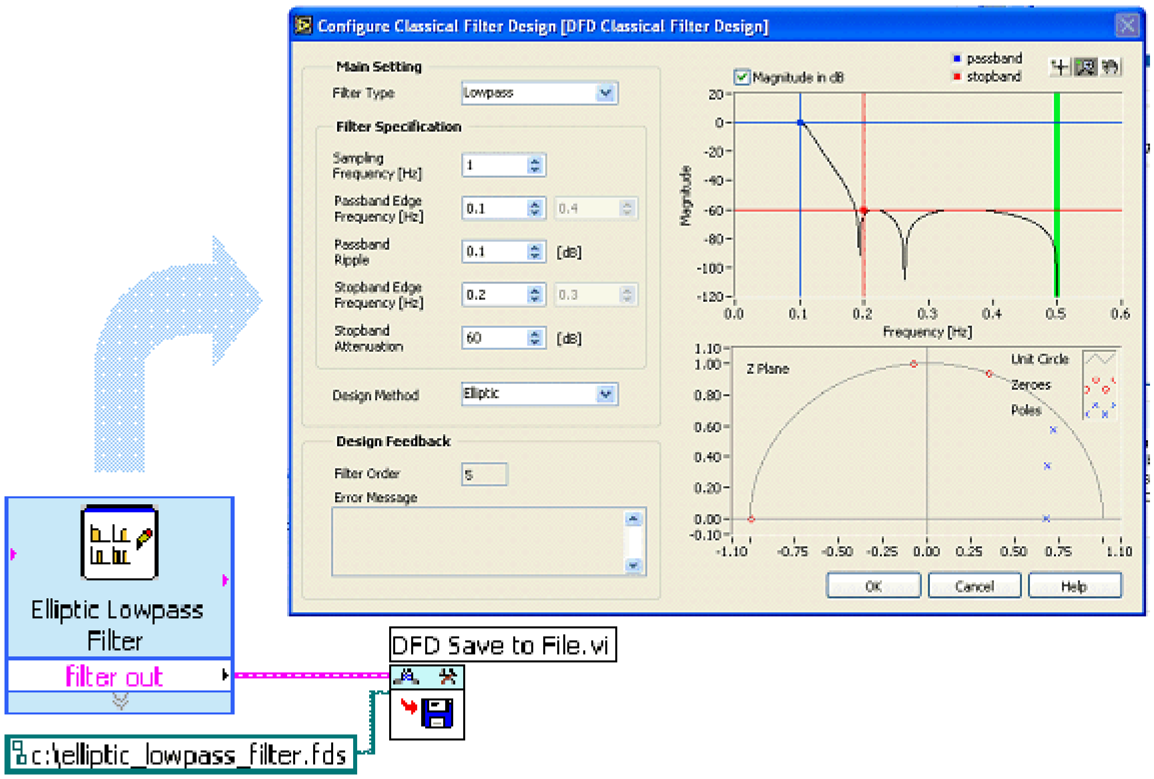
4.1 Graphical Programming for Signal and Image Processing

Signal and Image Processing is integral to biomedical engineering and is used in various courses[1,2,3]. Creating and manipulating complex signal and image processing algorithms textually is cumbersome and can lead to errors. Graphical programming provides a more intuitive approach to creating signal and image processing algorithms. To illustrate, let us consider the example of creating a filter. The parameters of the filter are shown in Table 1.

Table 1. Parameters for a Digital Filter

Criteria	**Value**
Filter Type	Lowpass
Cutoff Frequency	200 Hz
Filter Topology	Elliptic

Traditional techniques would involve generating the coefficients first in one program, use another program to redesign this filter if a fixed point target is being used, and finally using an add-on or separate program to download this filter to the hardware that is used. Additionally, the student does not get to explore the nuances of designing filters. Questions like, what would happen to the frequency response if some other topology was used? What is the order of the filter if we used an IIR configuration? How does my stop band and pass band characteristics change depending on the topology? Graphical programming, on the other hand, can help educators teach these concepts more effectively. Figure 7 shows an example of a filter design function that uses a configuration screen to design the filter.



Main Setting

Filter Type: Lowpass

Filter Specification

Sampling Frequency [Hz]: 1

Passband Edge Frequency [Hz]: 0.1 0.4

Passband Ripple: 0.1 [dB]

Stopband Edge Frequency [Hz]: 0.2 0.3

Stopband Attenuation: 60 [dB]

Design Method: Elliptic

Design Feedback

Filter Order: 5

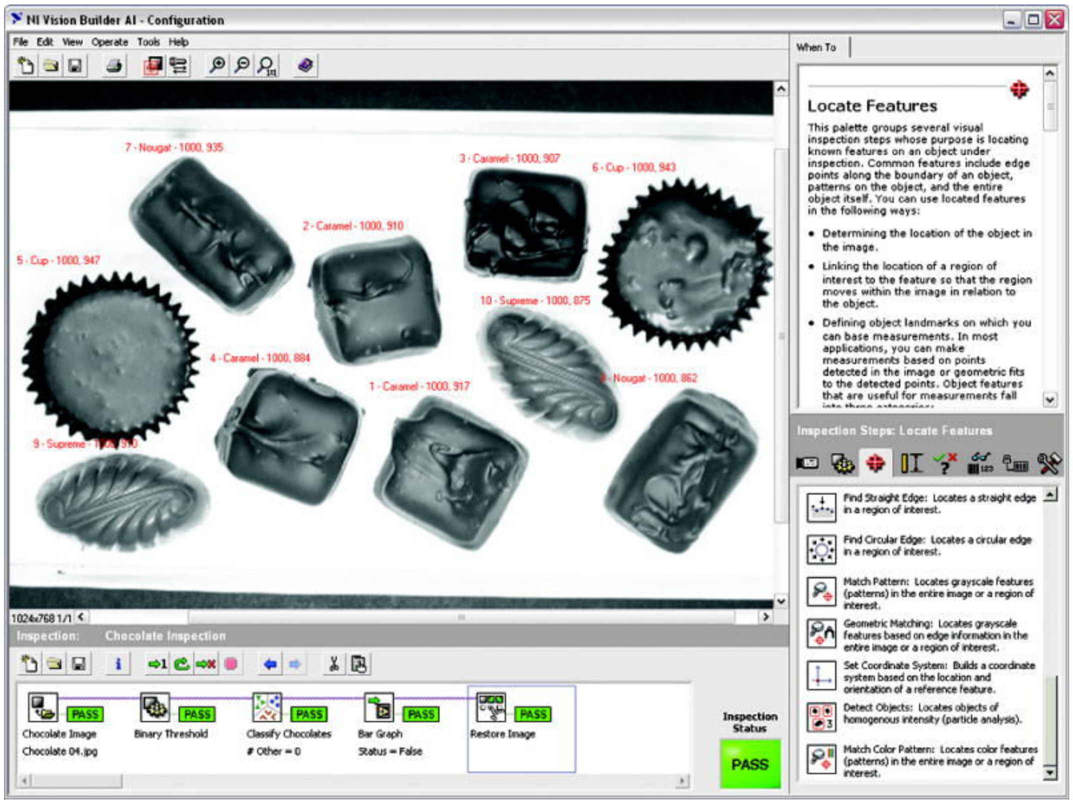
Error Message

Figure 7. Designing an Elliptic Low-pass Filter

Some key points about the configuration screen shown in figure 7 include dynamic update of the frequency responses and pole-zero plots. This implies that whenever the topology or any other parameter is changed on the left hand side, the right hand side will update automatically. In addition, educators can also demonstrate the changes in parameters by changing the frequency response on the right hand side. Once the filter is designed, the student may choose to save the coefficients to a file or convert for fixed point and generate code to implement this filter on an FPGA or DSP.

The key factor here is that since this is graphical programming, there is no additional learning curve for the student. Most packages also include a lot of signal processing functions that cover a wide array of signal processing functions.

A similar parallel can be drawn for image processing. Graphical programming environments allow for step-by-step creation of image acquisition and analysis programs. Educators and students can now acquire from many kinds of cameras from a USB webcams to high-end firewire cameras and use tools like the NI Vision Builder. Figure 8 shows an example program from Vision builder. It is worth noting that the system uses a set of functions that are added and configured in a step-by-step fashion – this allows the student to focus more on the concept of defect detection or object recognition than the task of writing the actual algorithm.



Chocolate Image: Chocolate 04.jpg — PASS
Binary Threshold — PASS
Classify Chocolates: # Other = 0 — PASS
Bar Graph: Status = False — PASS
Restore Image — PASS

Locate Features

This palette groups several visual inspection steps whose purpose is locating known features on an object under inspection. Common features include edge points along the boundary of an object, patterns on the object, and the entire object itself. You can use located features in the following ways:

- Determining the location of the object in the image.
- Linking the location of a region of interest to the feature so that the region moves within the image in relation to the object.
- Defining object landmarks on which you can base measurements. In most applications, you can make measurements based on points detected in the image or geometric fits to the detected points. Object features that are useful for measurements fall

Find Straight Edge: Locates a straight edge in a region of interest.

Find Circular Edge: Locates a circular edge in a region of interest.

Match Pattern: Locates grayscale features (patterns) in the entire image or a region of interest.

Geometric Matching: Locates grayscale features based on edge information in the entire image or a region of interest.

Set Coordinate System: Builds a coordinate system based on the location and orientation of a reference feature.

Detect Objects: Locates objects of homogenous intensity (particle analysis).

Match Color Pattern: Locates color features (patterns) in the entire image or a region of interest.

Figure 8. Example Program to Detect Chocolates in Vision Builder

4.2 Other Areas for Graphical Programming

Graphical Programming is continually evolving and now has languages capable of being used to teach control courses with complete simulation, system identification and control design capabilities[8]. Embedded design has also seen a resurgence in ease-of-use and focus on concepts with innovations such as the LEGO Mindstorms NXT kit[9] which is completely programmed with graphical tools. Graphical tools are already being used for Medical Device Design[10].While some of these courses may not be applicable today to biomedical engineering, they may well be relevant in the future with nanotechnology and BioMEMS.

5. Assisting ABET Accreditation

Because Graphical Programming involves using state-of-the-art technology and promotes hands-on project based learning with actual hardware, it is helpful in gaining ABET Accreditation[11]. With graphical programming, educators now have the opportunity to demonstrate hands-on learning of biomedical engineering concepts and designing real-world projects that students can solve in a team-based environment thus helping satisfy some of the essential ABET (a)-(k) outcomes[12].

6. Feedback on Using a Single Platform

Several educators from other disciplines such as Electrical engineering have reported on the effectiveness of both the graphical programming approach and using a tool that provides hybrid capabilities. Educators at Rose-Hulman Institute of Technology[14] have now started using graphical programming for teaching the signal processing classes and the results reported in literature are shown to be promising. Similarly, Educators at University of Texas at Dallas[13] have done studies on using a hybrid approach under a single platform and the results in literature are interesting and encouraging. While there are reports from other departments, the authors could not find any publication in the Biomedical engineering area. Being a novel concept, It is the hope of the authors that educators will leverage this approach and enlighten the academic community with the results

7. Graphical Programming – Tying it all together

It is important to note that what is being proposed in this paper is not to discard all the tools that are currently being used. In fact, Graphical Programming does not need educators to abandon and eliminate the current lab setups or tools. In fact, it augments the current system. They can continue using the existing software and hardware with the modified setup because Graphical programming provides interfaces for connecting to multiple buses, devices and software. For example, in LabVIEW, you can connect to various tools such as Code Composer Studio, Excel, Mathcad, Mathematica, Maple, Xilinx tools etc. What graphical programming brings to the table for educators is the possibility to use one platform and apply it to labs that include control and design concepts providing educators an avenue to have software that is economical and can be shared across multiple classes.

8. Conclusion

Biomedical engineering is a vast, continually evolving universe. Courses in biomedical engineering span multiple disciplines. This can lead to students learning too many different tools and not focusing enough on the concepts. In this paper, we discuss graphical programming, a novel approach to teaching concepts. Graphical programming languages such as NI LabVIEW and Ptolemy have evolved to span multiple courses and disciplines from bioinstrumentation to circuits, signal processing and embedded. Graphical programming offers an approach to teaching concepts where the programming tools serves as a means to the end, rather than the end themselves thus helping students focus more on the concepts. In addition, since graphical programming fosters project-based learning in a hands-on, team-oriented fashion, it helps with ABET accreditation.

7. References

[1] Johns Hopkins University BME Curriculum,
http://www.bme.jhu.edu/subsites/innovations/jhuinnovations.htm

[2] Duke University BME Curriculum
http://www.bme.duke.edu/downloads/undergrad_handbook.pdf

[3] University of Texas BME Curriculum
http://www.engr.utexas.edu/bme/ugrad/General%20Curriculum.pdf

[4] WM Johnston, JRP Hanna, RJ Millar, “Advances in dataflow programming languages “, ACM Computing Surveys (CSUR), 2004

[5] WH Ho, EA Lee, DG Messerschmitt, “High Level Dataflow Programming for Digital Signal Processing”, VLSI Signal Processing III

[6] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems," *Int. Journal of Computer Simulation*, special issue on "Simulation Software Development," vol. 4, pp. 155-182, April, 1994

[7] New Features in LabVIEW of Interest to Academia, http://zone.ni.com/devzone/cda/tut/p/id/3683

[8] Control Design Tools

[9] J.F. Kelly, LEGO Mindstorms NXT: The Mayan Adventure, ISBN: 1-59059-763-X

[10] Improving Retinal Disease Treatment with LabVIEW FPGA and Intelligent DAQ, M. Wiltberger, Optimedica Corp., http://sine.ni.com/csol/cds/item/vw/p/id/698/nid/124400

[11] ABET Website, http://abet.org/

[12] ABET (a)-(k) Program Outcomes 2007-08, http://abet.org/Linked%20Documents-UPDATE/Criteria%20and%20PP/R001%2007-08%20ASAC%20Criteria%2011-10-06.pdf

[13] N. Kehtarnavaz and V. Peddigari, Using hybrid programming in DSP lab courses, to appear in TI Developer Conference Proceedings, Dallas, March 2007

[14] “A Study of Graphical Vs Textual Programming for Teaching DSPs”, M Yoder, B Black - 2006 ASEE Annual Conference & Exposition