Journal of Signal and Information Processing, 2015, 6, 92-98
Published Online May 2015 in SciRes. http://www.scirp.org/journal/jsip
http://dx.doi.org/10.4236/jsip.2015.62009

Scientific Research Publishing

Toolkits for Real Time Digital Audio Signal Processing Teaching Laboratory

Kizito Nkurikiyeyezu, Faustin Ahishakiye, Cyprien Nsengimana, Etienne Ntagwirumugara

Department of Electrical and Electronics Engineering, University of Rwanda, Kigali, Rwanda
Email: k.nkurikiyeyezu@kist.ac.rw, f.ahishakiye@kist.ac.rw, c.nsengimana@kist.ac.rw, e.ntagwirumugara@kist.ac.rw

Received 5 February 2015; accepted 1 April 2015; published 2 April 2015

Abstract

This paper describes an audio digital signal-processing toolkit that the authors develop to supplement a lecture course on digital signal processing (DSP) taught at the department of Electrical and Electronics Engineering at the University of Rwanda. In engineering education, laboratory work is a very important component for a holistic learning experience. However, even though today there is an increasing availability of programmable DSP hardware that students can largely benefit from, many poorly endowed universities cannot afford a costly full-fledged DSP laboratory. To help remedy this problem, the authors have developed C#.NET toolkits, which can be used for real-time digital audio signal processing laboratory. These toolkits can be used with any managed languages, like Visual Basic, C#, F# and managed C++. They provide frequently used modules for digital audio processing such as filtering, equalization, spectrum analysis, audio playback, and sound effects. It is anticipated that by creating a flexible and reusable components, students will not only learn fundamentals of DSP but also get an insight into the practicability of what they have learned in the classroom.

Keywords

DSP, Filter, Digital Audio Effects, Graphic Equalization, Laboratory Teaching

1. Introduction

The main objective of engineers is to employ materials, energy, and information to create beneficial solutions for humankind. Thus, most engineering instruction should take place in laboratory to create engineers whose knowledge goes beyond mere theory [1]. A laboratory class is an excellent environment to teach and learn engi-

How to cite this paper: Nkurikiyeyezu, K., Ahishakiye, F., Nsengimana, C. and Ntagwirumugara, E. (2015) Toolkits for Real Time Digital Audio Signal Processing Teaching Laboratory. *Journal of Signal and Information Processing*, **6**, 92-98.
http://dx.doi.org/10.4236/jsip.2015.62009

neering since students are given opportunities to think and solve real world problem. This is particularly true in Rwanda as its government aims at making Rwanda's economy based on the knowledge of its citizens [2]. However, because of limited budget, Rwandan high learning institutions cannot often afford expensive full-fledged laboratories.

Due to these financial constraints, universities resort to simulation-based laboratories instead of hardware based laboratory experiments. For instance, at the University of Rwanda (UR), many engineering courses are supported by software-based simulation using MATLAB, LabVIEW, and NI Multisim. Additionally, through the iLab-Africa project, UR utilizes online laboratories (iLabs) to support introductory engineering courses. Via iLabs, real laboratories are accessed through remote facilities at Massachusetts Institute of Technology (MIT) and/or any other universities that use iLabs. This reduces the costs associated with laboratory equipment and greatly expands the range of experiments that students can perform in the laboratory. This simulation and computer-based learning give students the opportunity to see and have a certain feeling of how theoretical concepts learned in the classroom can be applied to solve real world challenges. Nevertheless, simulations do not allow students to interact with real laboratory equipment; thus, they cannot be considered equivalent replacements for hands-on laboratory.

In this paper, the authors discuss the design and implementation of modest digital audio toolkits that are developed to serve as an educational tool to allow students to learn basic real-time audio signal processing using any managed programming language. The main objective of these toolkits is to teach students the fundamentals of DSP through practical applications of the DSP theory and to familiarize students with modern programming techniques. The paper concludes with a discussion of possible use of the toolkit and several planned future improvements to the toolkits.

2. Existing Analogous Libraries

Given the availability of capable affordable computers, many DSP libraries have been developed. Several of these frameworks share many similarities with the toolkits described in this paper. For the purpose of this synopsis, only freeware libraries are surveyed.

On the NET Framework, the Exocortex.Dsp [3] is a popular C# library for complex number and FFT algorithms. It provides single and double precision complex number data types, complex number math library, and complex and real symmetric FFT.

The Synthesis ToolKit (STK) is another C++ open source library for audio signal processing. It was developed at Stanford University by Perry R. Cook and Gary P. Scavone. This library provides a set of algorithms and classes to facilitate rapid development of music synthesis and audio processing software [4]. The STK is designed to provide a cross platform, real-time, and platform-independent library with ease of use and to serve as an educational tool. The STK offers a collection of unit generators for filtering, inputs, and outputs, and algorithms for sound synthesis and effect, which can be used for research, teaching, performance and composition purposes.

The Sound Object (SndObj) [5] is an alternative C++ open source library, licensed under the Gnu Public License (GPL). The SndObj makes available functions and algorithms relating to sound synthesis, analysis and processing, sound file management and other various DSP functions such as FFT. It also supports multi-threating functionality and supports other languages like Python and Java. Unlike the STK, SndObj is not platform-independent. It depends on other third-party libraries and includes platform specific code.

NAudio is yet an additional open source .NET audio and Musical Instrument Digital Interface (MIDI) developed in C# by Mark Heath, with contribution from many other developers. NAudio was primarily designed because the Framework Class Library (FCL) in the .NET framework has poor support of audio handling [6]. NAudio provides a wide-ranging set of useful classes from which to construct audio application that play and record or manipulate audio files in some other ways.

The ICST DSP library [7] is another C++ library that provides a collection of C++ routines to help rapid development of audio processing and analysis applications. Unlike most libraries, it offers industrial-grades DSP algorithms for prototyping, testing and implementing real-time applications without the need of switching development environment.

Finally, Un4seen's Bass Audio Library is yet another multi-platform C++ library for audio processing. Its main purpose is to provide developers with powerful and efficient sample, stream, MOD music, MO3 music, and recording functions. It includes countless other features like add-on plugin system, MOD position, support

for AIFF files, and floating-point sampling. It is written by Ian Luck and is actively supported via its official online forum [8]. Bernd Niedergesäß developed Bass.Net, a .NET wrapper for the Un4seen Bass Audio Library and its add-ons [9]. This wrapper works with the .NET Framework to target windows ×86 or ×64 platforms. It can also be used with the Mono.Net Framework to target Linux or Apple's OSX platforms. It has also a special version—the Compact Framework—which supports development for mobile devices (e.g. for the iPhone and iPad development, the Mono Touch Framework might be used to this end).

3. Toolkits Objectives

With the availability of a plethora of highly efficient and reliable digital audio processing libraries, it is quite tough to justify the effort of reinventing the wheel. However, this toolkit is being developed because most of the existing libraries are not designed for teaching purposes. While there are libraries such as [4] and [5] that can be used in undergraduate laboratory courses, they have been designed with assumptions that do not apply at the University of Rwanda.

First, it seems several libraries are written in C/C++. Indeed, since most DSP applications are very computational intensive and time sensitive, C/C++'s pointers provide specific optimizations that cannot be matched by most other high level language. Unfortunately, at the University of Rwanda, students aren't familiar with C/C++. They are instead at ease with the Java programming language. Although it is possible to teach the compulsory concepts of C/C++ before the laboratory, this would be time consuming and less practicable since C/C++ is much complex than Java and requires a steep learning curve. Secondly, many of the aforesaid libraries are designed for rea world DSP application; thus, they often sacrifice simplify and ease of use to performance and extensibility. They make available powerful, but multifarious programming paradigms that are difficult for students to grasp. Thirdly, many libraries are designed for portability and platform independence; thus, do not generally provide any Graphic User Interface (GUI). Finally, the authors wanted software components that can be easily used to interface with MATLAB as it is provides many built-in DSP algorithms.

Over the course of the design of these toolkits, its authors had the following fundamental goals:

- Provide a framework for digital audio processing laboratory course that can be used by students with prior limited programming experience;
- Provide an easy to use DSP modular routines that can be used for educational purposes;
- To allow students to develop programming skills and be able to develop simple real world DSP software whose processing performance is not very critical;
- Allow easy extensibility;
- Familiarize students with modern programming techniques.

4. Choice of the Toolkit's Programming Language

The choice of the programming language for these toolkits depends on many factors. The authors wanted to use a modern Object-Oriented Programming (OOP) language that can be easily used by students at the University of Rwanda. After examining all available viable options, the author decided to use C#.NET.

The choice of C# occurred for numerous reasons. It was first attempted to write the toolkits using MATLAB. However, this idea was soon abandoned mainly because MATLAB is an interpreted language; as a result, it is not suitable for Real Time Signal Processing (RTSP) as it usually executes slowly than native executable program [10]. Further, whereas it would be preferable to developer a custom wrapper around existing C/C++ codes, normally, C# on the window platform, is as fast as C/C++ because the Just-In-Time (JIT) compiler—the compiler that converts the Intermediate Language (IL) the first time the program executes—, can make optimizations that C/C++'s pre-compiled programs cannot achieve. This is because the JIT can provide optimizations depending on the platform's processor. For illustration, the JIT would produces different optimizations depending on if the processor is, let's say, AMD, Pentium 4, multi-core Intel architecture, and in the later, if the machine supports Streaming SIMD Extensions (SSE) or not [11]. Another major advantage of C# over C/C++ is its automatic memory management through garbage collection. This makes C# particularly easy to use and less error-prone because the programmer does not have to be concerned with manually destroying unused objects. C# also allows access to low-level programming. Unlike Java, C# support non-type-safe code. This is possible because in most cases, the Common Language Runtime (CLR) oversees the behavior of the Microsoft Intermediate Language (MSIL) code, and would prevent any dubious operations. In this context, the garbage collector

(GC) freely organizes and condenses free resources. In a managed environment, the GC monitors the lifetime of objects and frees them when no part of the program references them. This makes sure that there is no memory leak. Thus, typically, programmers using the managed runtime environment do not have to worry about memory allocation and de-allocation. In addition, the managed environment presents many other advantages such as overflow inspection and type safety checking. Nonetheless, the managed runtime environment proves to be disadvantageous in some cases because the programmer is not managing the memory himself. Accordingly, in some exceptional cases when the programmer wishes to directly access low-level functionalities, he can do so with unsafe (no type checking, no overflow handling, etc.) blocks which provides fast and efficient code comparable to C/C++ pointers. This feature has been handy to us when implementing time-critical real-time DSP algorithms. Writing unsafe code, allows programmers to write code similar to native C/C++ within a C# program. C#'s unsafe blocks are also invaluable in our context because they allow to easily reusing existing C/C++ DSP algorithms with very few modifications. The robustness of Bass.Net API is another major reason C# was chosen as the programming language for these toolkits. As said, Bass.Net API is a managed wrapper around the Bass Audio Library and its add-ons. Bass Audio Library and its add-ons are written in C/C++; hence, they provide highly optimized digital audio processing algorithms. They are also unmanaged (or native code); thus, cannot be directly used in a managed code unless complex Platform Invocation Services (PInvoke) are used. Fortunately, the Bass.Net API can be used to hide much detail about the PInvoke and make it easy for students to use.

5. Design Methodology

These toolkits are primarily designed as a teaching and learning tool. Although many design methodologies were considered, designing a library on DSP rarely start from scratch, but typically reuses existing libraries. Our design is not an exception in this regard. It builds on top of many other third party libraries:

- Bass Audio Library [8] and its add-ons via its managed Bass.Net Library [9] are used to handle audio playing and recording functions;
- Exocortex.Dsp [3] and Math.Net Iridium [12] are used for FFT algorithms and filtering;
- Many other libraries were obtained through VS's NuGet Package Manager, a tool used to install and manage a wide range of packages and keep them up to date. For example, Ninject [13] is used as a Dependency Injection (DI) container, the Moq Framework [14] is used as a mocking toolkit to create implementations of interfaces to use in unit testing and various other libraries for error logging and ID3 tags reading.

Additionally, many design pattern and guidelines discussed in [15] and [16] have been taken into consideration to help design, what the authors believe, flexible, maintainable, and easy to use software components.

6. Architecture

The tool kits is organized in four different layers. Higher layers depend on lower layers but lower layers are ignorant of higher layers. This architecture provides many advantages as discussed in [17]. Application layering allows the separation of concerns by breaking it down into separate functional layers with little overlapping of functionality. Good layering, modularization and encapsulation help create a software that is both maintainable and pliable. The toolkits are organized into four different layers as shown in **Figure 1**.

The domain layer is the lowest layer. It provides classes, interfaces and methods that are used by other higher layers. It provides audio playback and recording, audio tag reading and exception handling and logging. The DSP layer makes available DSP algorithms for filtering, FFT algorithms, audio effects and audio spectrum analysis. Some of these algorithms are not fully implemented; instead, they are provided as interfaces and abstract classes that students will have to implement during a laboratory. The Graphic User Interface (GUI) layer provides a user interface for playing, recording and plotting audio spectra and waveforms. Finally, the Unit Test layer is used to verify the implementation of DSP algorithms and helps find any bugs. The Microsoft unit testing framework is used for this purpose.

7. Functionality Testing Results

To test the toolkits, a simple test application was developed. The application load audio files in the user's music folder reads their ID3 tags, and populate them in a list view. It then initializes default settings and configuration

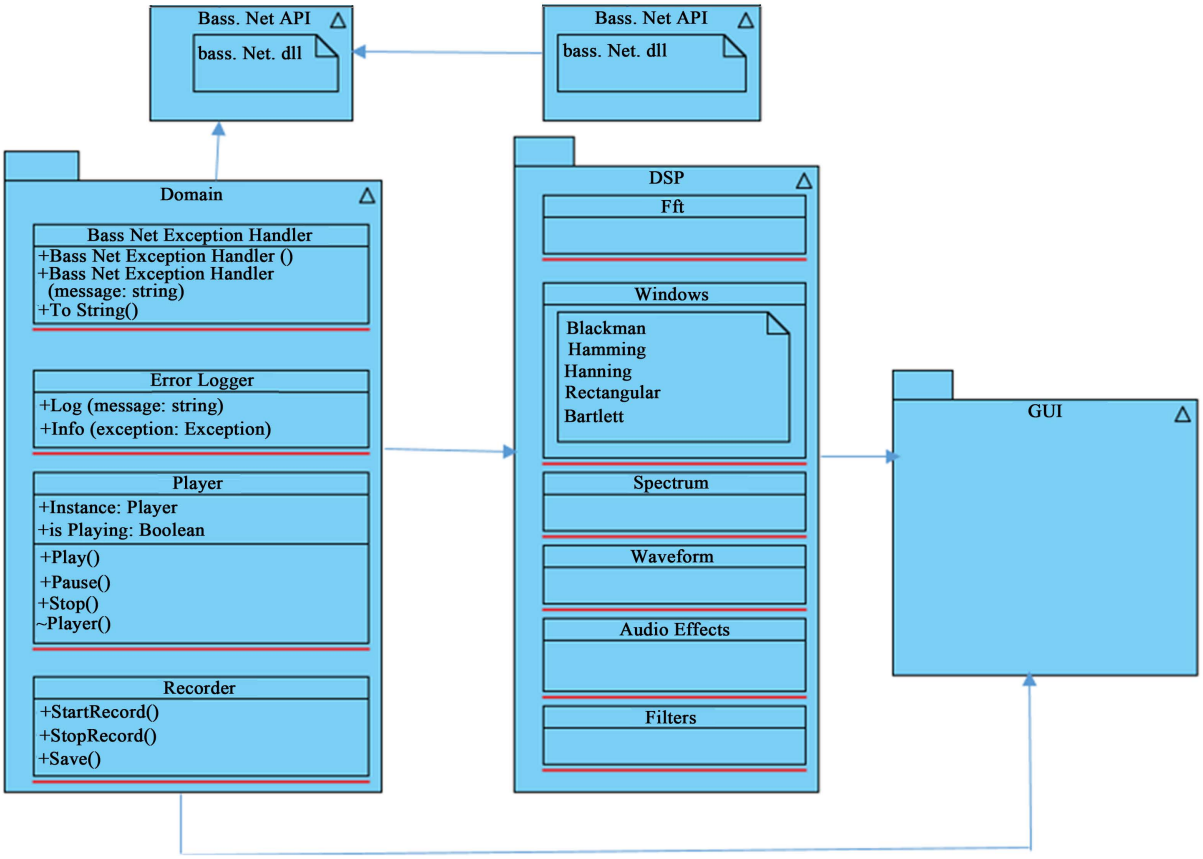

Figure 1. Layers of the toolkits with their major classes.

of the music player. At this time, the application showcases only fully tested module of the library viz. audio playback and audio ID3 tag reading, audio recording, and waveform plotting, and audio spectrum analysis. The screenshot of the running demo application is shown in **Figure 2**.

As of 2015, this toolkit has been used to teach two classes on DSP at the University of Rwanda. From a quick survey the authors conducted, the student overwhelmingly welcomed the added practical laboratory. They were particulary fascinated by its stunning GUI and the exciting audio processing application they were able to create. However, most students complained about the followings:

- Integration with MATLAB;
- The C# programming language proved to be quite challenging for some students;
- Lack of a comprehensive documentation;
- Lots of bugs;
- Signal plotting is quite "inflexible" and lack any customization.

8. Future Work

As discussed in the objectives of the toolkits, it is clear that what has been implemented is a very modest step in the development process. To address students' complaint and suggestions, many new features will be added in the future:

- Create .NET components from MATLAB to leverage MATLAB's powerful DSP algorithms and GUIs, which can be accessed just like any other native .NET components;
- Implement classes for spectrum analysis of sampled audio;
- Fully implement and test audio effects (Echo, Flanging, chorus, pitch change, etc.);
- Finish unit testing untested features;
- Implement a modular plotting mechanism to easily plot and display signal;

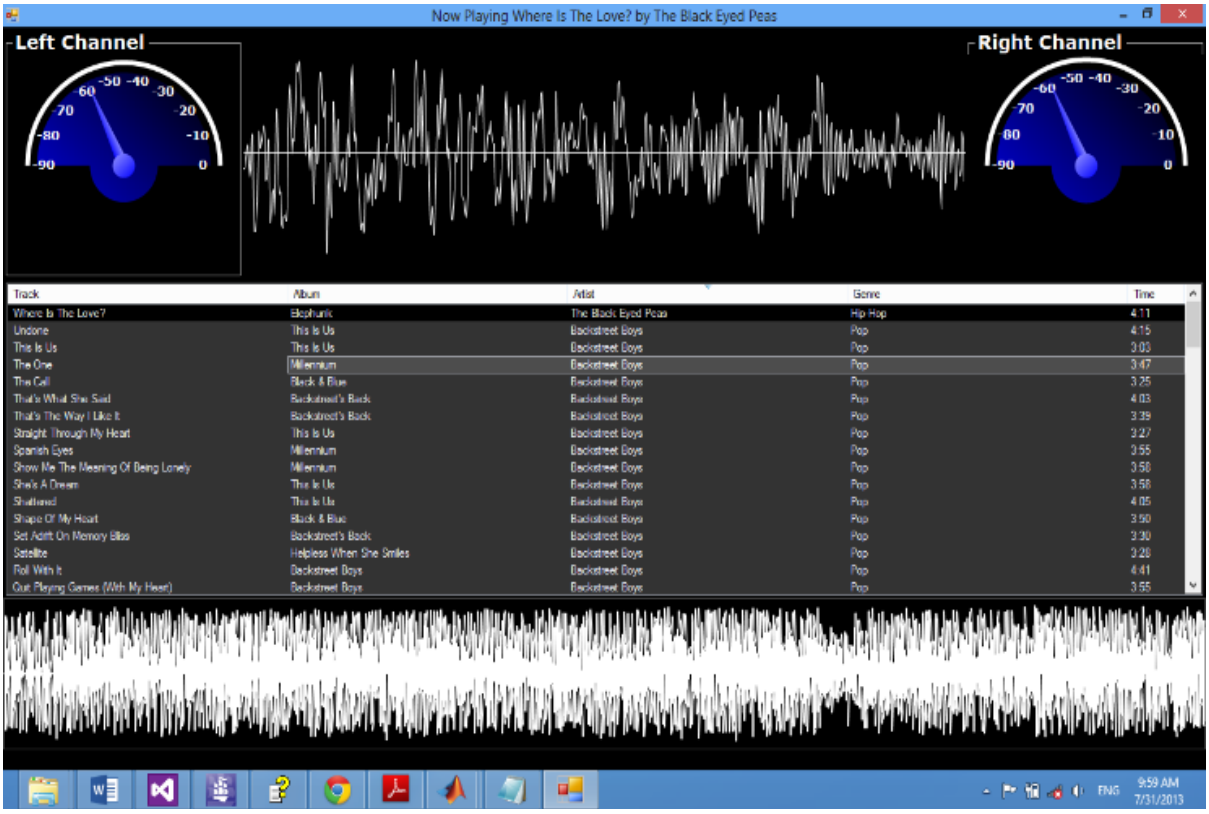

Figure 2. Screenshot of the running demo application.

- Do lot of refactoring and integration testing;
- Write a user manual and put together a sample laboratory description and provide any necessary "starter codes";
- Release an open source version of the toolkit;
- Algorithm optimization and testing (some algorithms are quite slow);
- Port the entire code of the toolkits to a much "easier" to learn programming language. After evaluating different programming languages choices, the Python programming language seems to be the most appropriate language for this endeavor. Since its creation, it has received numerous appraisal both from academicians and from professional in industry. Particularly, it is hailed for its easiness to learn and its productivity [18] [19].

9. Conclusion

Digital audio processing is sure to play an increasing role in the future for many applications in voice recognition, music recording, communication, and gaming. The .NET toolkit discussed in this paper is intended to offer an environment to get started with audio DSP and audio software development. It can be used with any managed programming language and provides commonly used components for digital audio processing. It is hoped that, through this toolkit, students will learn fundamentals of DSP through practice, that they will get an insight into the practicability of DSP and that they will develop life-long learning abilities.

References

[1] Feisel, L.D. and Rosa, A.J. (2005) The Role of the Laboratory in Undergraduate Engineering Education. *Journal of Engineering Education*, **94**, 121-130.

[2] Rwanda Ministry of Finance and Economic Planning (2013) Economic Development and Poverty Reduction Strategy (EDPRS). The Republic of Rwanda, Kigali.

[3] Houston, B. and Ptersen, B. (2003) Exocortex.DSP: An Open Source C# Complex Number and FFT Library for Mi-

crosoft .NET [Online]. http://www.exocortex.org/dsp/

[4] Cook, P.R. and Scavone, G.P. (1999) The Synthesis ToolKit (STK). *International Computer Music Conference*, 196-199.

[5] Lazzarini, V. (2001) Sound Processing with the SndObj Library: An Overview. *International Conference on Digital Audio Effect* (*DAFx*), **4**, 6-8.

[6] Heath, M. (2013) The NAudio Documentation Wiki [Online]. http://naudio.codeplex.com/documentation

[7] Papetti, S. (2000) The ICST DSP Library: A Versatile and Efficient Toolset for Audio Processing and Analysis Applications.

[8] Luck, I. (2005) Un4seen Developments - 2MIDI / BASS / MID2XM / MO3 / XM-EXE / XMPlay [Online]. http://www.un4seen.com/

[9] Niedergesäß, B. (2013) BASS.NET API for the Un4seen BASS Audio Library.

[10] Chapman, S.J. (2007) MATLAB Programming for Engineers. Cengage Learning, 3.

[11] Skeet, J. (2011) C# in Depth. 2nd Edition, Manning Publications Co., Greenwich, CT.

[12] Math.NET (2005) Math.NET Iiridium. http://iridium.mathdotnet.com/

[13] Anand, V. (2012) Dependency Injection Using Ninject. http://www.codeproject.com/Articles/424749/Dependency-Injection-Using-Ninject

[14] Moq: The Simplest Mocking Library for .NET and Silverlight. https://github.com/Moq/moq4

[15] Cwalina, K. and Abrams, B. (2008) Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable. Net Libraries, Addison-Wesley Professional, Boston.

[16] Hanmer, R. (2013) Pattern-Oriented Software Architecture for Dummies. John Wiley & Sons, Ltd., Hoboken.

[17] Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R. and Stafford, R. (2002) Patterns of Enterprise Application Architecture. Addison Wesley, Boston.

[18] Gaddis, T. (2012) Starting out with Python. Addison-Wesley, Boston, 1.

[19] Liang, D. (2013) Introduction to Programming Using Python. Prentice Hall, Englewood Cliff.