Bulletin of the *Transilvania* University of Braşov • Vol. 10 (59) No. 2 - 2017
Series I: Engineering Sciences

REAL-TIME SIMULATION SOFTWARE FOR ANALOG WAVESHAPING CIRCUITS

Sz. SIMON-BOGYÓ[1] M. CARP[2]

Abstract: *Even though a number of circuit simulation software exists, very few of them offer the functionality addressed in this paper. Which is not only to mathematically calculate the behaviour of a circuit with respect to virtual stimuli, but to use an input signal captured from the physical-realm and producing the circuit response in real-time. Two possible ways to create such an application are analysed: by modifying an existing open-source simulator and by creating one from the ground-up. It takes a close look at latency and afterwards proposing solutions on how to improve on this factor. Input latencies as low as 3 ms were obtained, which is below the human-perceivable delay, thus proving that real-time is something obtainable.*

Key words: *circuit simulation software, real-time, signal processing, C++, Java.*

1. Introduction

When faced with the task of implementing a waveshaping circuit, electronic engineers would rely mostly on experience, calculations and approximations done by hand or implemented on a SPICE-like circuit simulation software. But in order to obtain palpable results (e.g. when one wants to hear the „fuzz" effect produced by a guitar-pedal circuit) one must first implement the circuit as a prototype, connect it up to a testing rig and listen. Doing this is sometimes difficult, error-prone and costly in terms of time and money.

The power of simulating a circuit's behaviour before the actual implementation, consists in a great improvement in design efficiency, by highlighting faulty designs. Doing this as early on in the creative process as possible.

Thus, the authors tried to address the issue with a circuit simulation software capable of processing an audio input via a virtually designed waveshaping circuit, drawn up by the user of the software; thus greatly enhancing the testability of such circuits by providing the user with instant audible feedback. Doing this, by acquiring input from the built-in audio hardware of a PC. The software aims to be platform-independent making it thus a smart and scalable solution for professionals, students and enthusiasts alike.

[1] Master's student at the Electronics Systems and Integrated Communications Dept., *Transilvania* University of Braşov.
[2] Computer Science and Electronics Dept., *Transilvania* University of Braşov.

2. State of the Art

Investigation of the current state of similar software in this domain yielded that there are a limited number of such systems available commercially or open-source, and not one fully encompasses the full extent of what this paper proposes.

The software most similar to what this paper proposes is LiveSpice, which is a SPICE-like circuit simulation tool for processing live audio signals [6]. Having a simple to use and intuitive user interface mostly covering the goals of this research. It is the one of the two circuit simulation software which supports real-time audio input and output, the most important feature which this paper describes. The main drawback of this software is that there is limit to the complexity of the circuit being simulated.

Other two simulation software, LTSpice and Linear's PSpice are compatible with audio-domain processing but are not specifically intended for this. As both of them will process an audio file as input, thus, not being real-time solutions [7], [4].

BlockCompiler is another similar solution capable of efficient audio waveshaping. It is based on computational block objects and their interconnection networks, and it supports several different modelling paradigms. It is particularly powerful in creating physical models where two-directional interaction between physical elements has to be represented. High-level model specifications are compiled to efficient code, supporting real-time simulation and sound synthesis of relatively complex systems [8].

The presented application will process a real-time input acquired from the readily available hardware on the host PC thus being far more user-friendlier than [7], [4]. A specifically designed circuit solving solution will allow it to have a leading edge on [6] by making it more scalable and able to solve larger circuit diagrams. Being composed of actual circuit elements will render this software more useful for circuit designers and audio-circuit companies than BlockCompiler which is based on computational block objects rather than on electronic components.

3. Development

3.1. Prototyping

The first step taken towards the direction of a complete solution was to build a proof-of-concept model of what was desired.

In order to create this model, an existing open-source circuit simulation software was used, which could simulate small to medium size circuits in a very graphical fashion. The most important aspect of this simulator is that it already contained several circuit components such as programmable voltage sources and random signal generators which „injected" a stream of data into the designed circuit. Thus it was really simple to extend with an audio input element such that it could use it as a signal source instead of the aforementioned virtual sources. Basically, a new circuit component was added which used the Analog to Digital Converter (ADC) of the PC to obtain a real-life signal, and adapted it in such a way that the data could then be passed to the rest of the circuit. specified as references in the footnote (Insert Reference, Footnote, Bottom of page); these ones will be marked with Arabic figures. If several authors are part of the same organization, then its name may be written once and the authors will be marked with the same figure.

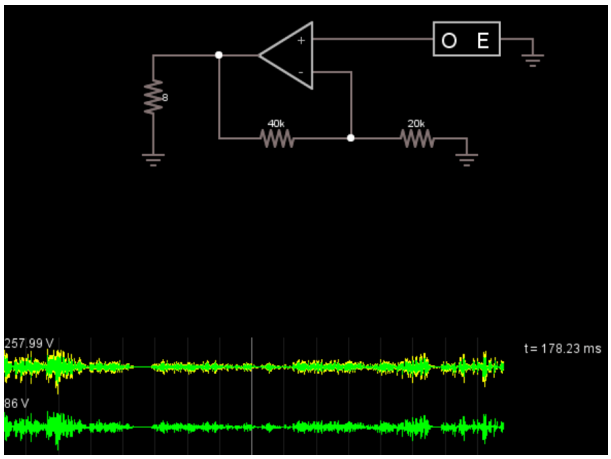

Fig. 1. *Real-life audio input processed by the circuit drawn in the prototype application*

In the image one can see the new input element, marked with „O E" supplying a simple schematic comprised of a non-inverting amplifier circuit and a resistor at the amplifier output. In the upper scope one can see the voltage across the 8 Ω resistor. In the lower scope one can see the voltage at the output of the microphone element (Note that the maximum voltage is displayed in the upper left corner of each scope, and is illustrative in this case, it being representative of an amplitude value retrieved from the audio card). It can be noted that the signal is not inverted and also amplified, proving that the idea works and is feasible to pursue further.

But, not far after implementing the prototype presented above, an issue arose which meant a shift in direction was needed: although it worked as expected, by amplifying the signal correctly, it was not fast enough to be used in real time. A large latency is observed between the moment of producing a sound, and seeing it's waveform on the scope of the simulator.

3.2. Audio Latency

In this sense, the quality of a real-time audio processing tool is tightly related to its latency. Which refers to a short period of delay (usually measured in milliseconds) between when an audio signal enters and when it emerges from the system [2].

Real-time voice transformations for instance should provide a sufficiently low latency so that the speaker or singer is not disturbed by the delay. While delays of less than 20 milliseconds are nearly unnoticeable with voice and most melodic instruments, delays as low as 4 milliseconds can be noticed with drum-like sounds [3].

Thus, latency is the main factor this paper tries to reduce in order to obtain a good quality product. In order to achieve this, an investigation was required.

Initially, one should ignore the hardware, assuming that the problematic bottleneck in not there: theoretically, an audio signal processor obtains a sample, which it then processes and plays the result. From a hardware point of view, this operation (but depending on the processing of course) should be trivial and should not produce any noticeable latencies. So, examine the system from a software point of view: First off, the

simulator presented before is written in Java and it was tested on a PC running Microsoft Windows 7 as operating system. The audio input element which was created, used the Java platform's standard sound API (application programming interface), JavaSound [9]. And the installed Windows audio driver was the standard built-in driver which came with the operating system.

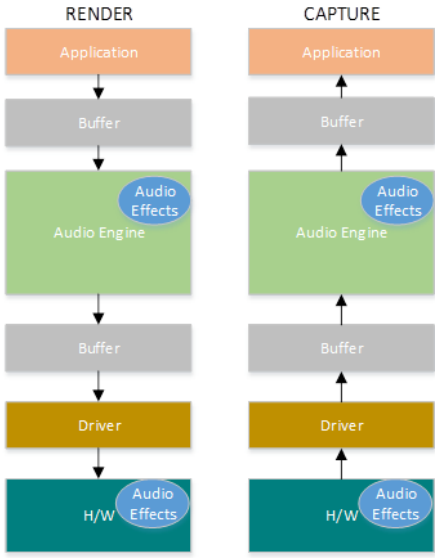

Fig. 2. *Microsoft Windows Audio Stack*

Looking at these facts from the ground-up, it can be noted that the standard Windows 7 audio stack, consists of a Driver, Buffer, Audio Engine and another Buffer (see Figure 2) each of which introduce a delay of its own. On top of these comes the application itself, which in turn contains another buffer, the processing itself, and Java-inherent operations which produce further latency. These inherent properties are the Java Garbage Collector (JGC) and the Java Virtual Machine's (JVM) so called HotSpot-Compiler.

In all of this, the Windows 7 audio stack introduces a round-trip latency of over 50 ms. (the Windows version is strictly specified, because Windows 10 brings a significant improvement to the audio stack latency) [10].

From the application's point of view, the interpreted nature of the language causes a significant processing impediment. When Java code is executed, it is not immediately compiled. Instead, when a loop is encountered for the first time, a few iterations are executed in interpreted mode, and then only, the code is compiled. This is the HotSpot feature of the JVM, which has been introduced since the release 1.4. While the speed of compiled Java code is fast enough for real-time computations, the first few iterations that are executed in interpreted mode can be too slow [5].

Another issue mentioned above is the Java Garbage Collector. More exactly, in order to complete its task, it will sometimes block all executing threads of the application regardless of their priority. And besides this, the amount of time during which these threads are suspended, is not known, nor guaranteed to be the same.

Initial experiments where done to to circumvent these platform and language issues using Java only (see "Real-Time, Low Latency Audio Processing in Java" [2]), turned out

to be cumbersome to implement. And would have meant to move the existing processing part into the Java Native Interface, which basically meant rewriting a large portion of the code in C/C++. This, leading to a large section of source-code being only "glue code" used to bind the separate modules together.

In the end, apart from the graphical interface, most of the business logic of the application would have had to be re-written, taking up probably more effort than writing everything from the beginning.

Taking all of the above in consideration, the authors opted to create a brand-new circuit simulation software, built from the ground-up with performance in mind.

3.3. A Simulator Built for The Purpose of Real-Time Audio Processing

As mentioned before, fixing the existing circuit-simulator written in Java, would have been a cumbersome and error-prone task which would not guarantee success.

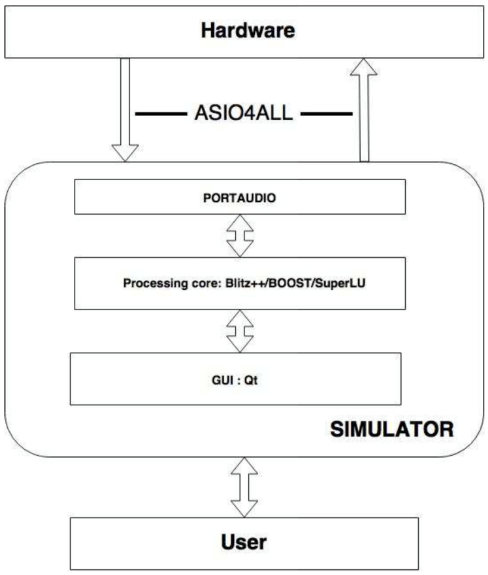

Fig. 3. *Schematic overview of the used libraries of the proposed simulator*

This is why, the best solution was to create a new piece of software which took the shortcomings noticed in the prototype and handled them appropriately. The new simulator is written in C++ only. It being a language described as „close to hardware", allows for great flexibility and excellent performance at the cost of an increased verbosity, which is still comparable to Java.

Using the same approach as in the previous sections, one may describe the new application from the hardware upwards. Using C++ for audio acquisition and processing, latencies of a few milliseconds can be achieved.

It is also worth mentioning that the problems faced earlier fade away since the code is always compiled into machine-code and there is neither garbage collector nor any virtual machine.

There are also quite a few available frameworks and drivers that readily support the low latencies described earlier. A few of which have been analyzed for the means of this work: PortAudio [11] and ASIO4ALL driver [12].

The schematic presented in Figure 3 describes how these external libraries are connected and used within the scope of the simulator. First-off, the ASIO4ALL audio driver connects to the hardware, and provides fast data transfer from the audio input source to the application.

ASIO bypasses the normal audio path from a user application through layers of intermediary Windows operating system software so that an application connects directly to the sound card hardware. Each layer that is bypassed means a reduction in latency (the delay between an application sending audio information and it being reproduced by the sound card, or input signals from the sound card being available to the application). In this way ASIO offers a relatively simple way of accessing multiple audio inputs and outputs independently [13].

The data provided by the ASIO driver is then accessed using the PortAudio API for C++. PortAudio provides a very simple API for recording and/or playing sound using a simple callback function or a blocking read/write interface. Example programs are included that play sine waves, process audio input (guitar fuzz), record and playback audio, list available audio devices etc.

On the other side, the user provides input to the simulator software by creating a schematic drawing of the circuit that will be used to process the incoming audio data. The GUI is created using the Qt framework [14].

Inside the application, the actual solving of the circuit drawn by the user is achieved by a separate processing core. This is achieved using three additional libraries:

- Blitz++, a mathematics library which specializes in matrix operations.
- SuperLU, a solver for linear equation systems.
- Boost, a huge collection of libraries, out of which the mathematics library was used.

4. Current State of Development and Results

After a few weeks of implementing the first results were obtained. The screen-capture seen in Figure 4 shows a circuit containing a few basic elements. Looking over the image, one can instantly notice that there are two main parts within the UI (user interface): the

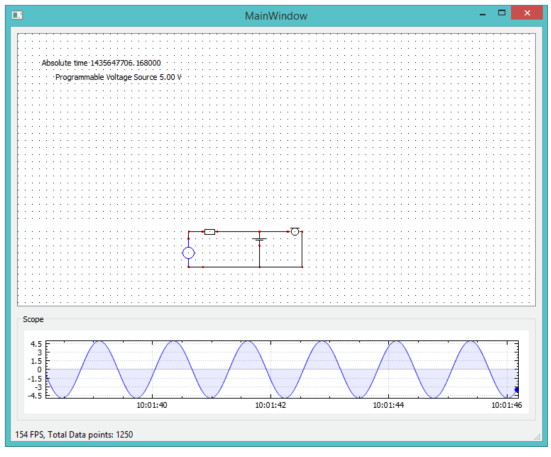

Fig. 4. *Screen capture of the C++ simulator*

working area, and the scope. In the working area the user may create his circuit, all circuit elements being available from the right-click context menu within the grid.

The figure presents a small electronic network created for testing purposes containing (from left to right), a programmable voltage source, a resistor, a DC voltage source and a microphone. Selected elements show up in blue (the programmable voltage source in this case). Right-clicking on an element produces a menu, which allows the user to remove the element, modify its parameters or visualize its voltage in the scope. In Figure 4, the scope contains the output of the programmable voltage source (the blue slowly oscillating sine-wave).

The data acquisition done by the microphone uses the ASIO4ALL driver, as discussed in the previous sections, this driver together with PortAudio enables us to measure the latencies from the various input devices listed in the figure.

The following list shows the minimum latencies, depending on the driver:

- Microsoft Sound Mapper - Input Latency: 0.090000
- Microphone (Realtek) Latency: 0.090000
- Microphone (ASIO) Latency: 0.003000
- Microphone (Realtek HD) Latency: 0.010000

Standing out from the rest, the ASIO driver's latency is the lowest at 3 ms.

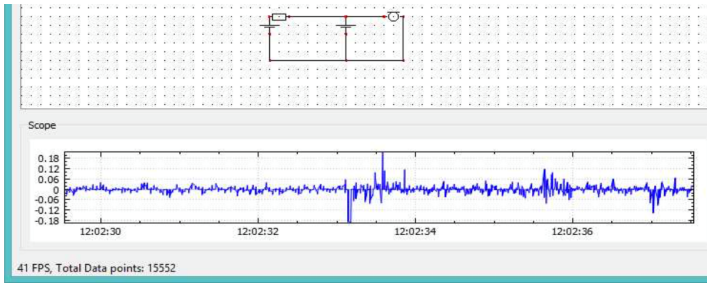

Fig. 5. *Noise captured by the microphone element and plotted in the scope of the C++ simulator*

Besides the previously presented functionality, the application aims to be a fully functional circuit simulation software with an intuitive and minimalistic user-interface.

5. Future Development and Conclusions

By further tweaking the settings of the used libraries and by optimizing the existing code, even better latencies may be reached. The presented application was tested on Windows 8.1. Since Windows 10, Microsoft has announced a new audio stack, which might offer better performance than ASIO and PortAudio used in our software, by promising a latency reduction of up to 16 ms [10].

Currently the software supports a few basic circuit components, but the palette of circuit elements can be vastly improved. Similarly, the method used for circuit solving is quite primitive. Literature now offers a large variety of solving techniques and algorithms [1], which could further improve performance.

The end result of development, could comprise out of the circuit simulator presented in the earlier sections plus a hardware module. The hardware module would contain a

programmable DSP chip, and the software would be extended in such a manner that, after the user designs his audio circuit, a transfer function would be computed for it and translated into code for the DSP. Basically meaning, that one would obtain a graphical integrated development environment for designing audio processing hardware. The hardware module would be programmed via USB and it would contain an audio input and output. The output would be processed by the circuit designed and uploaded to the hardware module by the user.

In conclusion, the paper presented the research done in the direction of achieving a truly real-time analog circuit simulation application. In this respect, numerous prototypes were implemented, which amount to a single proof-of-concept application which is able to show that the initial goal is attainable. Ultimately, use of such software would greatly improve prototyping times by moving a substantial part of trial and error into the simulation realm. With this, improving on initial hardware prototype's quality and reducing the product creation cost. Because of this, continued work in this direction would be relevant.

References

1. Fijnvandraat, G., Houben, S.H.M.J., et al.: *Time Domain Analog Circuit Simulation*. In: Journal of Computational and Applied Mathematics **185** (2006) No. 2, p. 441-459.
2. Juillerat, N., Arisona, S.M., et al.: *Real-Time, Low Latency Audio Processing in Java.* In: Proceedings of the International Computer Music Conference, Copenhagen, 2007, p. 1-4.
3. Lago, N.: *The Quest for Low Latency*. In: Proceedings of the International Computer Music Conference, Miami, Florida, 2004.
4. Lithwhiler, D.H.: *Listening to PSpice Simulations with LabVIEW*. In: International Journal of Engineering Education **21** (2005) No. 1, p.19-25.
5. http://scribblethink.org/Computer/javaCbenchmark.html. Accessed: 19.05.2017.
6. http://www.livespice.org/#overview. Accessed: 16.05.2017.
7. http://tny.im/aSZ. Accessed: 16.05.2017.
8. http://www.aes.org/e-lib/browse.cfm?elib=12597. Accessed: 19.05.2017.
9. http://java.sun.com/j2se/1.5.0/docs/guide/sound. Accessed: 18.05.2017.
10. http://tny.im/aSY. Accessed: 18.05.2017.
11. http://www.portaudio.com. Accessed: 18.05.2017.
12. http://www.asio4all.com. Accessed: 18.05.2017.
13. http://tny.im/aS-. Accessed: 19.05.2017.
14. http://www.qt.io/developers. Accessed: 19.05.2017.