Csound Instruments On Stage

Alex Hofmann
Institute of Music Acustics
University of Music and
Performing Arts Vienna
Vienna, Austria
hofmann-alex@mdw.ac.at

Bernt Isak Wærstad
Department of Music
Norwegian University of
Science and Technology
Oslo, Norway
bernt.warstad@ntnu.no

Kristoffer E. Koch
Independent Electrical
Engineer
Trondheim, Norway
koch@kristofferkoch.com

ABSTRACT

Low cost, credit card size computers like the *Raspberry Pi* allow musicians to experiment with building software-based standalone musical instruments. The COSMO Project aims to provide an easy-to-use hardware and software framework to build Csound based instruments as hardware devices. Inside the instrument, the Csound software is running on a *Raspberry Pi* computer, connected to a custom designed interface board (COSMO-HAT) that allows to connect potentiometers, switches, LED's, and sensors. A classic stomp box design is used to demonstrate how Csound can be brought on stage as a stand-alone hardware effect instrument.

Author Keywords

Csound, Raspberry Pi, Linux, Live Performance, Stomp Box, Digital Effects

ACM Classification

H.5.5 [Information Interfaces and Presentation] Sound and Music Computing, H.5.2 [Information Interfaces and Presentation] User Interfaces—Haptic I/O.

1. INTRODUCTION

Computer music software that allows musicians to combine ready-made building blocks for sound processing motivates them to design their own instruments. Especially in contemporary music, composers, interpreters, and improvisers are often using software instruments built with tools like Max/MSP, SuperCollider, PureData or Csound for live performances.

In the line of these software tools, Csound is unique due to the following features: a) Csound has a long tradition as a sound renderer software. Being written in portable "C programming language" in 1986 by Barry Vercoe, its roots go back to the tradition of the Music-N languages developed by Max Mathews at Bell Laboratories in 1956/57 [4]. b) The development of the Csound language is under the directive of a 100% backward compatibility, allowing to open and play Csound files written more than 30 years ago, even today with Csound 6.06 [4]. This feature also makes Csound a good choice in terms of stability and consistency, as all current code will be guaranteed to also run in the future. c) Furthermore, Csound offers an API for C, C++, Python, HTML, Java and more [9, 5]. This makes Csound a powerful audio renderer for various applications, ranging from sequencer software (Blue[1]), over a VST-Plugin engine (Cabbage[2]), an engine for iPad Apps[3] (iVCS 3[4]), a game sound engine (Csound Unity[5]) and a sound playback engine for smartphones[6]. d) Finally, Csound is a LGPL licenced Open-Source software which can be installed, used, and adapted freely for all use cases and runs on most of the common operating systems (e.g., Windows, MacOS, Linux, Android, iOS). e) Additionally, Csound has a vast amount of free documentation and learning tutorials [6][7].

For live performances, audio software is mostly used on personal computers and laptops. Although laptops allow to take a customized software-based setup on stage, disadvantages are a) the need to connect external controllers and audio interfaces, which is time consuming on stage, and b) a laptop solely dedicated for live performance is expensive. Laptops are mostly used for other tasks as well, which require the installation of multiple software packages and drivers on the same system. This may influence the stability of the live performance setup.

Through the availability of credit card size, cheap (35$) linux-based computers like the Raspberry Pi (RPI) it is possible to build low-cost, software-based, stand-alone musical instruments [3, 2]. For such a setup, the operating system can be optimized for real-time audio processing.

The Csound Euro Rack Module is taking this approach by setting up an Arduino Board, a Raspberry Pi and Csound so that the signal processing power of Csound can be integrated into analog modular synthesizer systems [8]. An installation package of Csound for the RPI is available via apt-get [1] or Csound can be compiled from the source code[8]. Running Csound on a RPI needs an environment for input and output (I/O) of audio signals and for controller inputs to Csound.

In the current paper we are presenting a hardware and software framework called COSMO (Csound on Stage Musical Operator) which facilitates to build Csound-based standalone instruments.

NIME'16, July 11-15, 2016, Griffith University, Brisbane, Australia.

[1] `http://blue.kunstmusik.com/`
[2] `http://cabbageaudio.com/`
[3] `https://github.com/csound/csound/tree/develop/iOS`
[4] `http://csound.github.io/showcase/2015/11/22/ivcs3/`
[5] `http://rorywalsh.github.io/CsoundUnity/`
[6] Find an overview of Csound applications under `http://csound.github.io/create.html`
[7] Find tutorials and documentation under: `http://csound.github.io/documentation.html`
[8] To compile the latest Csound Version for the RPI follow the build instructions on github: `https://github.com/csound/csound/blob/develop/BUILD.md#raspian`

2. MOTIVATION

The idea for the COSMO project started in 2013 on the Linux Audio Conference in Graz when Edgar Berdahl gave a workshop on *Making Embedded Musical Instruments and Embedded Installations using Satellite CCRMA*[3, 2] and we discussed possibilities to run Csound-based effects [7] on such a device.

Stomp-boxes are a popular design for effect engines in a live setup. Predominantly used by guitarists, stomp boxes are also often part of other live-electronic setups used by keyboarders or DJ's. Although alternative interface designs are under development [10], musicians (especially guitarists) are used to turn effects on and off with their feet and to modify the effect settings with potentiometers and switches on the box. In our "basic design", Csound is running on a RPI in a stomp box case as a stand-alone device (see Figure 1 and 3).

3. INTERFACE DESIGN

As digital signal processing is very powerful and versatile, it is often combined with traditional analogue instruments and analogue hardware. Hereby, it is a challenge to combine the analogue/acoustic domain with its classic interfaces, musicians practised on for years and know the playing techniques, with the new, often changing and non-idiomatic interfaces coming from the digital domain. Our COSMO design attempts to provide an easy integration of custom digital signal processing into existing analogue hardware effect chains, i.e.stomp box guitar pedals. A custom designed interface board (COSMO-HAT), attached to the RPI allows to connect up to eight analogue controller inputs, eight digital controller I/Os and MIDI-I/O sockets (Figure 2). A benefit with this design is the dedicated hardware which makes it less prone to be unstable.

We do not provide a standard layout for the COSMO interface in terms of the number and position for the buttons and the potentiometers. This allows the musicians to design an interface for their specific needs. A guitar player might want to change parameters with his feet and probably wants a different layout than a vocalist who has both hands available during a live performance. Figure 1 shows three different designs created by participants in a COSMO workshop. The size of the stomp box and the limitation in the number of controllers may be seen as a limitation, but also forces the user to work more thoroughly with the mappings of controllers to the software.

A USB sound card inside the stomp box provides I/O of the audio signals. Another feature of our design is a built-in analogue cross mixer between the dry and the processed signal. This helps to preserve the original tone of the input signal and allows to use COSMO for send effects with longer latency and larger buffer sizes.

3.1 COSMO-HAT

The COSMO-HAT is a "Hardware Attached on Top" for the RPI. It is a printed circuit board that contains a micro controller (Atmega 1284P, by Atmel) to a) read-out up to 8 control inputs (potentiometers); b) control up to 8 LED's or button switches. The board is open source[9] and designed in KiCAD[10], an open source CAD tool for electronics. We have also made room for connecting MIDI I/O, and interfacing the versatile WS2812 "neopixels" (by Adafruit) a protocol to control multi-color LEDs.

[9]All materials can be found under `https://github.com/cosmoproject`

[10]`http://kicad-pcb.org/`

Figure 1: Three designs for COSMO (Csound on Stage Musical Operator) stomp boxes by workshop participants.

The firmware (the software running on the Atmega 1284P micro controller on the COSMO-HAT) is open source, and communicates with the RPI over a serial connection (SPI). That will allow for extension of features, e.g. offloading real time tasks to the micro controller, like on an Arduino. One example of offloading tasks is a digital noise filter, currently running on the micro controller. The firmware for COSMO-HAT is provided as an executable script on our pre-configured RPI (Raspbian) operating system available as a download. No external tools or cables are required for the communication between the COSMO-HAT and the RPI, including all updates of the firmware.

The COSMO-HAT is fully integrated into the python script provided with the project resources (including a pre-configured Raspbian image with examples). This script manages the communication between the csound engine running on the RPI and the COSMO-HAT. With this design the user can directly start to create Csound instruments using only the Csound language.

3.2 COSMO Cross mixer

For the design of a COSMO effect pedal, we included a true bypass circuit, that allows the performer to turn off the device immediately and output the clean input signal. The COSMO cross mixer is a relay based stereo true bypass switch and a dry/wet potentiometer. True bypass means that when bypass is enabled a) no signal will be coming from the COSMO effects b) no signal will run into the COSMO inputs. Another feature is, that if power is lost, the COSMO cross mixer mechanically falls back to bypass mode. The cross mixer is a fully analogue circuit, which works without any delays, providing a fail-safe interface for the sound operator[11].

3.3 Software

COSMO is designed to run Csound for audio processing at first hand. A patch layout to map the controllers of the box to the parameters in Csound is provided together with some example effects[12]. Through the history of Csound, thousands of Csound instruments covering most of the existing sound synthesis and sound modifying techniques exist[13]. This makes Csound a suitable software platform for such an application.

[11]The schematic of the Cross mixer circuit can be found online: `https://github.com/cosmoproject/bypass_crossmix/blob/master/bypass_crossmix-sch.pdf`

[12]`https://github.com/cosmoproject/cosmo-dsp`

[13]For inspiration check out Iain McCurdys 'Real Time Instruments': `http://iainmccurdy.org/csound.html` or the classic collection of Csound instruments `http://www.csounds.com/resources/audio/`.

Figure 3: COSMO in a chain of guitar effect pedals.

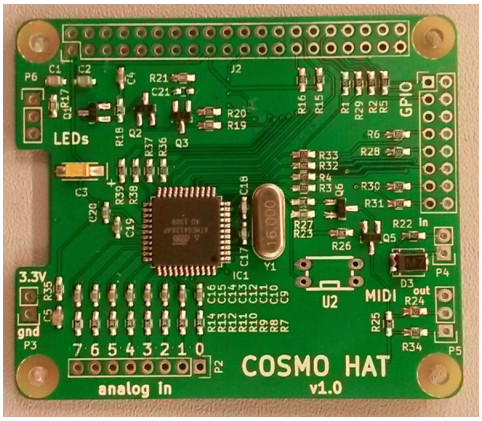

Figure 2: COSMO-HAT circuit board to connect 8 analogue controller inputs, 8 LED's or switches and MIDI I/O.

3.4 Audio Interfaces

Depending of the choice for an audio interface one has to find a compromise between the flexibility of the design and the input-output latency. The usage of USB-audio interfaces with the RPI allows to use any linux supported interface[14]. An other option to interface sound to the RPI is the cirrus logic audio card (CLAC) which is especially designed for high quality audio I/O and is connected to the GPIO header of the RPI[15].

We tested audio input-output latency (I/O) for a USB audio interface (UCA-222, by Behringer) and the CLAC with Csound on a RPI2 Model B (900MHz quad-core ARM Cortex-A7 CPU, 1GB RAM). A "RimShot" sound was played with a TR-8 drummachine (by Roland). The sound was split with a VLZ 1402 Mixer (by Mackie). One signal went directly into a Focusride audio interface and the other signal was routed via an "Aux Send" of the mixer into the RPI and then into the Focusrite audio interface. Both signals were recorded simultaneously with Ableton Live, using a sampling rate of 44,000 Hz (16 Bit). The delay between both signals was calculated.

With the UAC-222, the smallest hardware buffersize we were able to run on the RPI without artefacts was 512 samples (-B512 -b64; ksmps = 64)[16]. This resulted in a I/O latency of 30 ms. The same setting with the CLAC showed half the latency (15 ms). Moreover, we were able to reduce the hardware buffer to 256 samples (-B256, -b32; ksmps = 32) which resulted in 7 ms I/O latency.

Unfortunately, the GPIO pins used in current COSMO-HAT design overlap with the GPIO pins used by the CLAC. In future designs we aim to support the CLAC by allowing to jumper the COSMO-HAT to different GPIO pins.

4. WORKSHOP CONCEPT

The COSMO framework was designed as a basis for workshops on *how to build stand-alone music instruments based on Csound*. With the current framework, the duration of a workshop requires at least two full days to build the device. This includes, soldering all connection pins on the COSMO-HAT, creating a knob layout, drilling holes into the enclosure, soldering the connections from the knobs to the COSMO-HAT, extending the network and USB extension from enclosure to RPI, and running a basic Csound FX instrument[17]. Apart from assembling COSMO in a workshop, detailed build descriptions are given on the COSMO website (`http://cosmoproject.github.io/`), which allows to build the device independently.

5. FUTURE WORK

COSMO is a hardware and software framework which is designed to build a stand-alone Csound-based effect instrument. The framework allows for adaptation and modification of the "basic" effect pedal design. Connecting a MIDI keyboard would allow to build a Csound based synthesizer,

[14] All USB 1.1 audio class compliant devices are supported by linux. Futhermore, a list of supported professional audio interfaces can be found under `http://wiki.linuxaudio.org/wiki/hardware_support`

[15] An easy way to install the required real-time kernel for the cirrus logic audio card is via `http://rpi.autostatic.com/`. Add the repository by following the descriptions on the website. Then *apt-get* the following packages: linux-image-cirrus, cirrus-config-overlay, cirrus-config-modprobe, cirrus-config-scripts, cirrus-config. All information is from the element14 forum: `https://www.element14.com/community/thread/31714?start=45&tstart=0`

[16] Optimizing I/O latency with Csound `http://www.csounds.com/manual0LPC/UsingOptimizing.html`

[17] See building steps `http://cosmoproject.github.io/docs/`

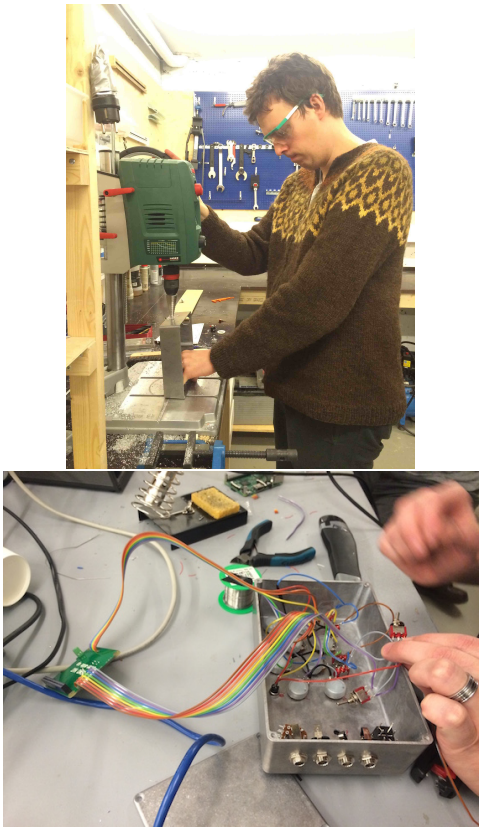

Figure 4: Participants designing their own interface panel, by drilling holes for the buttons, LED's, audio sockets, and switches into the enclosure and soldering the connections to the COSMO-HAT.

other sensors (e.g., photo resistors) or more audio outputs might be a suitable setup for a sound installation. The COSMO workshops are meant to give novices an easy introduction into building their own stand-alone digital effect instruments, with a primarily focus on sound design in Csound. No programming knowledge is required, as all components are complementary. However, with the current framework, we noticed that an advantage of not providing a fixed interface layout encourages the creativity of the participants on one hand, but on the other hand also makes it a more time consuming procedure, than simply assembling a fixed number of pre-designed parts. A possible solution to this might be, that in the workshop, only a very basic (one button, one LED) stomp box is built first and if there is time remaining, the participants can expand their designs individually. From this consideration a redesign of the COSMO-HAT is foreseen, which is based on using one (or multiple connectable) small and less expensive I/O board (COSMINI-HAT). Each board is responsible for only one controller I/O instead of the big COSMO-HAT with multiple I/O's. With this method the design and the price of a COSMO box is resizeable. Overall, the COSMO framework is a suitable basis to undertake workshops in *designing and building digital instruments based on Csound* with musicians having more experience in sound design than in programming. An even simpler *COSMO-light* framework is forseen for the future which also allows to work with high school students. Here, the making steps will need less time consuming soldering of components and might be based on the 5$ *Raspberry Pi Zero* and the COSMINI-HAT.

6. ACKNOWLEDGMENTS

This research was supported by the Arts Council Norway.

7. REFERENCES

[1] P. Batchelor and T. Wignall. Beaglepi. *Csound Journal*, 18, 2013.

[2] E. Berdahl. How to make embedded acoustic instruments. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2014.

[3] E. Berdahl and W. Ju. Satellite ccrma: A musical interaction and sound synthesis platform. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2011.

[4] R. Boulanger. *Ways Ahead: Proc. of the First International Csound Conference*, chapter Csound Past, Present and Future: Keynote for the Opening of the First International Csound Conference, pages 2–8. Cambridge Scholar Publishing, Newcastle upon Tyne, UK, 2013.

[5] Gogins. Composing music for csound in c++. *Csound Journal*, 17, 2012.

[6] J. Heintz, A. Hofmann, and I. McCurdy, editors. *Csound - Floss Manual*. FLOSS Manuals Foundation, first edition, 2011.

[7] A. Hofmann, A. Mayer, and W. Goebl. Towards a live-electronic setup with a sensor-reed saxophone and csound. In *Linux Audio Conf.*, pages 153–156, 2013.

[8] I. Ikenberry and J. Lim. Csound eurorack module. *Csound Journal*, 18:1–9, 2013.

[9] V. Lazzarini. *Ways Ahead: Proc. of the First International Csound Conference*, chapter User-Developer Round Table I: The Technology of Csound, pages 24–31. Cambridge Scholar Publishing, Newcastle upon Tyne, UK, 2013.

[10] S. Suh, J.-s. Lee, and W. S. Yeo. A gesture detection with guitar pickup and earphone. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2014.