KES Transactions on Innovation in Music: Vol 1 No 1
Special Edition - Innovation in Music 2013 : pp.147-157 : Paper im13bk-014

The OWL: an Open Source, Programmable Stage Effects Pedal

Thomas Webster[1], Guillaume LeNost[2], Martin Klang[3]

[1]*University of Bedfordshire, Luton, LU1 3JU, UK*
thomas.webster@beds.ac.uk
[2]*Lionfish Audio, London, SE4 2PB, UK*
guillaume@lionfishaudio.com
[3]*Rebel Technology, London, E2 8HD, UK*
martin@rebel-it.com

1. Abstract

The Open Ware Laboratory (OWL) is an Open Source, programmable stage effects pedal that takes a step towards putting the power and flexibility of computer-based audio Digital Signal Processing (DSP) into dedicated hardware devices and instruments for performing musicians. The OWL provides an Application Programming Interface (API) that makes it possible for users to program their own audio effects applications, or 'patches' for the device in the C++ programming language. This paper introduces the concept and motivation behind the OWL and outlines the fundamental differences in approach between it and other similar hardware audio processing devices.

2. Introduction

Electric guitar / stage effects pedals have been in existence since the 1960s (the 'Maestro Fuzz Tone Pedal' from 1962 is widely thought of as the earliest example), and the use of these is widespread amongst electric guitarists as a way of enhancing the natural sound of the instrument and exploring different sonic palettes unavailable through the guitar and amplifier alone. Effects pedals are also popular with other instrumentalists, electronic musicians in general, and in recording studio contexts as external effect processors when mixing-down [1, 2].

Electric guitar effects pedals can be distinguished in some ways from multi-effects stage pedals, in that they can be thought of generally as single-effect devices, which can be made entirely with analogue electronics. Multi-effects pedals typically use digital technology to allow the stage musician to be able to choose from a selection of pre-programmed digital audio effects, or combinations of such [3]. Using digital technology vastly reduces the required footprint for complex effects processing, and digital multi-effects units can have a large number of different audio DSP effects to choose from.

An important difference with the OWL is that it allows the user to program bespoke digital audio effects in a simple API using the C++ programming language [4]. So,

although the device is closer to the second category of digital multi-effects pedals, the OWL goes further and permits the development of entirely unique, user-designed DSP processing, rather than being limited to a selection of pre-set effects afforded by the manufacturer. Due to the Open Source [5] nature of the project, and that it uses a popular programming language, there is also potential to build a community of users and programmers around the pedal, allowing people to freely share their effects patches and code online.

3. Existing Devices

The most obvious precedents for this work are traditional guitar and stage multi-effects pedals, but this section will also look at some other programmable hardware systems, which are also relevant to our project.

There are a couple of digital audio effects pedals currently available that exploit the widespread use of mobile devices for the distribution of software. The Digitech iStomp [6] and Zoom MS100-BT both allow the downloading of ready-made effects patches using online software distribution systems such as the Apple App Store.

The iStomp system from Digitech, allows users to download software digital audio effects (called e-Pedals) from the Apple App Store onto their iPhone and then transfer the effect from their iPhone to the pedal hardware via a proprietary cable (iPhone connector to DIN variant). In the same vein, the more recent Zoom MS100-BT uses a Bluetooth connection and does away with the proprietary cable dependency of the Digitech device. Although both of these systems afford the possibility of an ever-growing library of reasonably priced digital audio effects, neither can be programmed directly by the end user.

The Line6 ToneCore [7] pedal range takes a different approach to programmable DSP hardware for guitarists. Each guitar pedal comprises a base unit and one of a range of interchangeable hardware effect modules that plug in to the base unit. Each plug-in module has up to six knobs, a printed label and a Micro Controller Unit (MCU) that contains the DSP code for that specific effect. The base unit contains a Freescale Symphony DSP chip, which runs the program from the plug-in module. Various plug-in modules are available covering a range of different audio effects. More relevant to our project is the ToneCore developer kit, which consists of a special base unit with a Universal Serial Bus (USB) socket for connection to a Windows PC, and a programmable module. This system allows the user to program effects on their computer, and then load the code onto the MCU residing on the programmable module, which can then be connected to the base unit in the same way as a standard module. There are some limitations with this system in that the user has to program in assembly language that is specific to the Symphony chip, and the tools required to program the chip are proprietary, which makes it difficult to for a programmable device using this chip to be Open Source. In addition, because the company need to sell the hardware plug-in modules, it makes it commercially difficult for Line6 to distribute code for their stock effects

without undermining their own market. Additionally, the ToneCore can only be programmed from a Windows PC, unlike the OWL, which has a cross-platform API. Probably the closest device to ours is the OpenStomp [8] guitar pedal, a project started in 2007, and arguably the first Open Source programmable guitar effects pedal. The pedal uses a Propeller Parallax microcontroller, which can be programmed with a combination of a proprietary high-level byte coded language called 'Spin' and a low-level assembly language. So, although programmable, like the Line6 ToneCore developer kit, it relies on a chip that has to be programmed with a specialist proprietary system.

Other devices and systems that have been developed recently are related to the OWL as programmable hardware rather than as effects pedals or stomp boxes. Snazzy FX's Ardcore [9] module for modular synthesizer systems is based around an Arduino [10] compatible chipset, which is programmable in C using the Arduino Integrated Development Environment (IDE). The Ardcore is able to do some limited audio processing, but only has an 8-bit converter, which is not high enough resolution for good quality audio applications; the OWL, in comparison has a 24-bit converter. The Ardcore module is useful for low fidelity audio and controlling the behaviour of other synthesizer modules by generating and manipulating Control Voltages. For instance, the module could become a bespoke sequencer, LFO or envelope function.

Another system that developers are adopting for creating bespoke audio devices is the Raspberry Pi (RPi) [11], a computing platform based around an ARM11 chip that runs an embedded Linux Operating System (OS), a common choice for many devices including mobile phones and car computers [12]. An advantage of running an operating system such as Linux is that users can choose from a range of already available software. Some audio projects on the RPi have used Pure Data [13]; a widely used, Open Source visual programming language, but equally users could program in any language / environment supported by Linux.

The OWL, in contrast has firmware with no dependencies, which will always be able to run effects code written in C++ providing it is written in the required format. In addition, where the RPi is a multi-purpose computer capable of performing many different functions, the OWL is a dedicated device designed for a particular use (live music performance). An advantage of this is that, because the pedal is optimised for audio processing only, the device becomes simpler to design and maintain, with fewer dependencies on third-party activity.

4. Design

The OWL pedal software is designed to be easily programmable, and the hardware modifiable by users with a little experience of working with electronics. The device is aimed squarely at a demographic consisting of performing musicians, in particular guitarists, who are interested in hardware and software 'hacking'. The name OWL is an acronym of Open Ware Laboratory and this refers to the fact that the entire project is Open Source in both hardware and software; all of the

technical information and code relating to the project is freely available under the Gnu General Public License (GPL).

Although not without precedent, the OWL differs from most of the guitar effects pedals mentioned above in its approach to the idea of making the whole architecture as accessible as possible. To this end, the OWL API is entirely cross-platform, and C++ is used for effect patch development. C++ has the advantage of being one of the widely used programming languages, with compilers available for most computer architectures and operating systems. Using the API, patches can be developed with a simple text editor, makefile and compiler such as GCC, or alternatively using an IDE such as XCode or Visual Studio.

4.1 Hardware

The electronics in the pedal have been split between two circuit boards, partly for considerations of space (fitting all of the components on one Printed Circuit Board (PCB) would have resulted in a larger footprint), but also to keep a degree of electrical segregation between the analogue and digital components. The hardware is designed to be modifiable so that end users could potentially house the PCB's in a different chassis and add more controls to create a completely different hardware audio device. Figure 1 below shows the final revision of the OWL stage effects pedal.

Connections: The OWL has five ¼" jack connectors on the pedal comprising left and right input and output and expression pedal input, allowing for the development of stereo patches and foot control of parameters. On the rear of the pedal is a power input socket for a 9v DC power supply and a micro USB socket for connection to a computer or MIDI device. Using the USB On-The-Go [14] protocol, the OWL pedal can be used as either a USB host or peripheral device, which allows for the device firmware and patches to be updated from the computer, and also enables MIDI peripherals to be connected over USB, allowing for remote control of the OWL (and expanding the amount of potentially controllable parameters) from an external MIDI controller.

Hardware Controls: On the pedal are four freely assignable potentiometers, which can be programmed to control any particular functionality within your effect patch. Also on the top is on the top is an illuminated pushbutton with a bi-coloured LED, used to select between two separate memory slots, each of which can store an individual effect patch. Alternatively, the user could program a single patch containing several effects (or chains of effects), and use one of the assignable knobs to switch between them. The pedal also features a true stereo analogue bypass footswitch which bypasses all of the circuitry on the pedal itself and passes the signal present at the input directly through to the output jacks.

Figure 1. The OWL stage effects pedal.

Analogue Circuit Board: The analogue circuit board uses all thru-hole components, and also connects to the hardware controls on the top of the pedal. The audio and expression pedal jack sockets are mounted directly to the PCB, which simplifies assembly by obviating the need for wire harnesses. The inputs and outputs are buffered to ensure high input impedance and low output impedance, which allows for a range of different audio devices other than guitars to be connected to the inputs. Also on the board is a two-pole active low-pass filter before the ADC with a cut-off frequency of 28kHz to help prevent aliasing artefacts. Lastly, the analogue board also houses a voltage regulator and power conditioning components to allow the pedal to be reliably powered from either USB or an external power supply. PCBs are assembled in-house.

Digital Circuit Board: The digital circuit board houses the MCU, which performs all of the processing tasks on the pedal, an external codec which handles all signal conversion between the analogue and digital domains, and an 8Mbit external Static Random-Access Memory (SRAM). All of the components on this board are Surface Mount Technology (SMT). The digital and analogue sections of this board have isolated ground and power planes, which are spread out over four layers. Due to

the small pitch of some components the PCBs have a gold immersion finish and are machine assembled.

Microcontroller: The Microcontroller Unit used for the OWL is an ARM Cortex M4-based STM32F4 variant, a powerful processor with a speed of 168MHz and capable of 210 Dhrystone Million Instructions per Second (DMIPS). The chip has 192Kb RAM and 1Mb flash memory and also features a Floating Point Unit (FPU), Direct Memory Access (DMA) and integrated DSP. Traditionally, dedicated DSP chips have been preferred for audio processing as they tend to have better performance, but the Cortex M4 series can come very close to matching the performance of DSP chips [15]. Some of the devices listed in section 3 use dedicated DSPs, and in the case of the Line6 Tonecore and OpenStomp effects pedals, it could be argued that, although the use of these chips allows for high performance, because they require the use of specific assembly code, they are more difficult to program than an ARM microcontroller. As many features of the ARM are programmable in C++, this makes it simpler to develop the firmware, and also the software framework to support new effects patch development in C++. The ARM Cortex M4 could be classified as a Digital Signal Controller (DSC) as it is essentially a microcontroller with DSP features and multiple memory buses, and the additional DSP features were a factor in choosing this processor. Lastly, and just as important is that an Open Source tool chain is available with GCC / ARM, enabling the whole project to remain Open Source under GPL.

External Codec: The codec used for analogue to digital conversion and vice-versa is the Wolfson WM8731, a stereo codec capable of from 8 to 96kHz sampling rates in 16 to 24 bits resolution and a 90dB signal to noise ratio. The WM3871 features programmable analogue input and output gain stages, allowing for a wide range of sensitivity and headroom settings. The codec features oversampling at 250 times the sampling frequency and has built in digital anti-aliasing filters for high quality conversion, and combined with the low-pass filter on the analogue board input stage, makes for a good anti-aliasing system.

Memory Expansion: The built-in microcontroller memory of 192Kb is complemented by external 8Mbit static random-access memory (SRAM). The SRAM is 16 bits wide and is very fast with 10nS access times. It connects to the flexible static memory controller (FSMC) bus of the ARM chip, which can access it with minimal wait states, ensuring good memory performance.

4.2 Software

The OWL currently has three downloadable software applications related to it - OwlNest, OwlSim and OwlWare, and also a library of effects patches, OwlPatches [16].

OWLNest: OwlNest is an application built to communicate with the OWL pedal hardware. The user can control sample rate, bit depth and audio input and output levels via the Graphical User Interface (GUI). From here they can upload patches

to the two available memory slots on board the OWL, selecting from any of the `.hpp` effects files currently in the OwlPatches folder and added to the `includes.h` and `patches.cpp` files.

Both OwlSim and OwlNest contain a copy of the OwlPatches folder and reference this as a library of patches. When creating new patches or downloading from the online patch library, these can simply be added to the folder and the `includes.h` and `patches.cpp` files, and this will add them to the patch registry.

OwlSim: OwlSim is a software framework designed for developing effects patches for the pedal. Based on the JUCE [17] C++ library, and using the Steinberg Software Developers Kit, the framework allows the user to build Virtual Studio Technology (VST) plugin versions of their patches, which can be useful for designing and debugging with a Digital Audio Workstation (DAW) or VST host.

The code listed in Figure 2 below is the main `Stompbox.h` file, which all patch files must include as they inherit from the Patch class in this file. `Stompbox.h` also includes the `AudioBuffer` class shown which is used for passing audio samples in and out of the effects patch. The other virtual functions listed in the Patch class are methods to determine the amount of samples processed in each block and sample rate of an external host program. The methods relating to parameters are used to register and name the four controls on the pedal and to read values from them. Most of the work is done in the `processAudio()` function, which takes a type `AudioBuffer` as an input and is where the sample by sample audio processing takes place.

The patch class is deliberately stripped down as much as possible and has only a minimal set of functions, providing audio input and output and access to the four hardware potentiometers.

OwlPatches: OwlPatches is a folder consisting mostly of individual `.hpp` files of effect patches that can be loaded onto the pedal. Also in the folder is an `includes.hpp`, file which essentially just lists the header files for inclusion at compile time, and another file which lists the patches to be registered with either the simulator or for use on the pedal.
Creating an effects patch to build as a VST, or to be compiled for the pedal is relatively easy. The code snippet in Figure 3 shows a template patch with the bare minimum needed for a patch to run. Although there is no audio function to this code, there is an example of how to register one of the knobs on the pedal and also the `processAudio()` function for the sample processing loop.

```
#ifndef __StompBox_h__
#define __StompBox_h__

#include <string>
class PatchProcessor;

enum PatchParameterId {
  PARAMETER_A,
  PARAMETER_B,
  PARAMETER_C,
  PARAMETER_D,
  PARAMETER_E,
  PARAMETER_F
};

class AudioBuffer {
public:
  virtual ~AudioBuffer();
  virtual float* getSamples(int channel) = 0;
  virtual int getChannels() = 0;
  virtual int getSize() = 0;
};

class Patch {
public:
  Patch();
  virtual ~Patch();
  void registerParameter(PatchParameterId pid, const
std::string& name, const std::string& description = "");
  float getParameterValue(PatchParameterId pid);
  int getBlockSize();
  double getSampleRate();
  AudioBuffer* createMemoryBuffer(int channels, int samples);
public:
  virtual void processAudio(AudioBuffer& output) = 0;
private:
  PatchProcessor* processor;
};

#endif // __StompBox_h__
```

Figure 2. Stompbox.h file.

```
#ifndef __TemplatePatch_hpp__
#define __TemplatePatch_hpp__

#include "StompBox.h"

class TemplatePatch : public Patch {
public:
  TemplatePatch(){
    registerParameter(PARAMETER_A, "My Knob");
  }
  void processAudio(AudioBuffer &buffer){
    // put your code here!
  }
};

#endif // __TemplatePatch_hpp__
```

Figure 3. Template file for effect patch development.

Figure 4 shows an example volume control patch. It is a trivial example but serves to show how little programming work needs to be done in order to create a working audio processing patch with the provided framework. There are only five lines of code added to the template patch in order to read data from a control into a variable, then iterate over the samples from the buffer and multiply them by the value from the registered control.

```
#ifndef __GainPatch_h__
#define __GainPatch_h__

#include "StompBox.h"

class GainPatch : public Patch {
public:
  GainPatch(){
    registerParameter(PARAMETER_A, "Gain");
  }
  void processAudio(AudioBuffer &buffer){
    float gain = getParameterValue(PARAMETER_A);
    int size = buffer.getSize();
    for(int ch=0; ch<buffer.getChannels(); ++ch){
      float* buf = buffer.getSamples(ch);
      for(int i=0; i<size; ++i)
    buf[i] = gain*buf[i];
    }
  }
};

#endif // __GainPatch_h__
```

Figure 4. Simple volume control patch.

Currently, the OWL has a library of commonly used guitar effects which features patches including overdrive, modulation effects like phasing and flanging, delays, reverbs (including a port of the Open Source reverb, 'Freeverb') and some combination effects (e.g. delays with modulation and filters). Also in the current library are some interesting synthesis patches including a 'Dub Siren' synthesizer by Charles Verron and 'Drone box', a sympathetic resonating synthesizer by Oli Larkin. These two patches demonstrate the potential of the OWL as a tool for synthesis and sound generation.

5. Conclusions

Overall, this has been a successful project, and although not without precedent, the authors believe that the OWL has broken some new ground. By providing an API for a dedicated hardware audio device, which can be programmed in a commonly used programming language (C++), the OWL lowers the bar for entry into programming embedded audio devices.

Although opting for a C++ API certainly makes it easier for the end user to program the functionality of the pedal, some feedback from potential users of the OWL

suggests that ideally they would like the API to be made easier still. An option for future development along this line could be to develop a lightweight, dedicated textual coding IDE for the OWL project, without the vast range of options available for more advanced programmers that can be found in the XCode and Visual Studio IDEs, for example. This would negate the steep learning curve encountered with this kind of software development environment, and would allow the user to simply focus on the audio DSP programming alone. There is also a range of possibilities that could lead to simpler programming interfaces for the pedal, such as working with existing software platforms like Pure Data or Faust [18].

Furthermore, there are also plans to develop other kinds of programmable hardware audio devices using the platform we have developed, and to introduce the project to the academic environment as a learning tool for audio DSP programming.

6. Acknowledgements

We would like to thank the London Music Hackspace, all at ROLI, Andrew McPherson, Max from SFX, and Torsten Anders for his support in preparing this paper.

7. References

[1] Izhaki, R.: Mixing Audio: Concepts, Practices and Tools, p.458. Taylor & Francis Ltd. (2008)
[2] Case, A.U.: Sound FX: Unlocking the Creative Potential of Recording Studio Effects, p.336. Taylor & Francis Ltd. (2007)
[3] Hunter, Dave.: Guitar Effects Pedals: The Practical Handbook, p.7. Hal Leonard (2004)
[4] Stroustrup, B.: The C++ Programming Language. Addison-Wesley Publishing Company (1985)
[5] Perens, B.: Open sources: voices from the open source revolution. The open source definition. p.171–185. O'Reilly Media, Inc. (1999)
[6] iStomp | DigiTech Guitar Effects, http://digitech.com/en-US/products/istomp.
[7] Line 6, http://line6.com/tcddk/.
[8] OpenStomp | The World's First OpenSource Digital Guitar Effects Pedal, http://howleraudio.com/frontpage/.
[9] SNAZZY FX, http://snazzyfx.com/ardcore.html.
[10] Arduino - HomePage, http://arduino.cc/.
[11] Raspberry Pi, http://www.raspberrypi.org/.
[12] Puckette, M.: Pure Data: another integrated computer music environment. Proceedings of the Second Intercollege Computer Music Concerts. pp. 37–41 (1996)
[13]Dalheimer, D.K., Welsh, P.: Running Linux. p.730. O'Reilly Media, Inc. (2006)
[14]USB.org - USB On-The-Go, http://www.usb.org/developers/onthego/.
[15]pingdynasty/OwlPatches, https://github.com/pingdynasty/OwlPatches.

[16]Beckman, P.: AES - Real-time Audio Processing Capabilities of Microcontrollers, Application Processors, and DSP's [Online]. (2011). Proceedings of the 131st AES convention, New York, 2011. Available from:http://www.dspconcepts.com/sites/default/files/white-papers/2011%20AES%20-%20DSP%20vs%20Micro%20rev%202.pdf [Accessed: 15 November 2013].
[17]About JUCE | JUCE C++ Library, http://www.juce.com/about-juce.
[18]Home, http://faust.grame.fr/.