Proceedings of the 18th Linux Audio Conference (LAC-20), SCRIME, Université de Bordeaux, France, Nov 25–27, 2020

PD-FAUST MACKIE CONTROL

Albert Gräf

IKM, Music-Informatics
Johannes Gutenberg University (JGU) Mainz, Germany
aggraef@gmail.com

ABSTRACT

The paper describes faust-mcp, a Pd abstraction which interfaces Faust to control surfaces utilizing the Mackie Control Protocol (MCP). It builds on the author's Pd-Faust software which enables you to run dsp programs (such as synthesizers and effects) written in Grame's Faust programming language inside Pd. The add-on can be used to control Faust dsps in Pd using MCP-compatible controller hardware and software.

1. INTRODUCTION

Grame's Faust is a functional programming language which greatly facilitates the programming of audio processing and instrument plugins [1]. Faust programs can be compiled to native code for an abundance of different signal processing environments and plugin standards. Pd-Faust is a plugin which allows Faust programs to run in Miller Puckette's graphical real-time patching software Pd[1]. It offers dynamic loading (and reloading) of Faust modules, MIDI[2] and OSC[3] control and sequencing, as well as automatic GUI generation (in the form of graph-on-parent subpatches), cf. [2].

Faust dsps typically offer a number of different controls for various parameters, such as the cutoff frequency and resonance of a filter, oscillator and envelope parameters of a synthesizer, etc. These are represented in a Faust program by means of so-called UI (user interface) elements, cf. [3]. While Pd-Faust lets you generate Pd GUIs for all UI elements of a Faust dsp in an automatic fashion, it is often desirable to control such parameters by means of some external, physical control surface instead. To these ends, Faust lets you map MIDI messages to each UI element by corresponding meta-data in the UI element specifications. For instance:

```
res = hslider("res [midi:ctrl 20]", 3, 0, 20, 0.1);
cutoff = hslider("cutoff [midi:ctrl 21]",
                 6, 1, 20, 0.1);
```

The controller mappings are in the square brackets following the control names. The `midi:ctrl tag` specifies the kind of MIDI message to be received by the program, in this example CC20 for the resonance and CC21 for the cutoff control, respectively. These input values can then be used in the Faust definition of the dsp as needed, e.g., for computing the required filter coefficients. UI elements for output (so-called bargraphs) are available as well, and all of these can be mapped to different kinds of MIDI messages (pd-faust only supports MIDI CC bindings at this time, however).

[1] `http://puredata.info/`
[2] `http://midi.teragonaudio.com/`
[3] `http://opensoundcontrol.org/`

So Faust dsps can already be controlled by plain old MIDI controllers with a few knobs or faders quite easily, by just adding a small amount of meta-data to the Faust program. However, this method quickly becomes unwieldy when using a lot of different Faust programs in the same patch, since most MIDI fader boxes won't easily accommodate a large amount of parameters, and remapping the controls is often a tedious task. Thus some form of automatic mapping of the controls is needed, and you also want to be able to quickly switch between different banks of controls.

As luck would have it, this kind of functionality is readily provided by so-called DAW (digital audio workstation) controllers, and there is an established MIDI-based protocol for these, the Mackie Control Protocol (MCP). This is what faust-mcp uses to interface pd-faust to compatible controllers. In the paper, we give a quick introduction to MCP, discuss how the faust-mcp package utilizes it, illustrate faust-mcp's usage with an example, and finally discuss some future work to further improve the interface.

2. MACKIE CONTROL

DAW controllers were invented to ease the operation of digital audio workstation (DAW) software [4]. They often resemble a mixer control surface, which seems sensible because mixing is a big part of what a DAW program does, and most musicians and studio engineers will be well familiar with that kind of interface.

Thus DAW controllers typically have a number of faders and knobs used to input track parameters such as volume, panning, sends, etc., along with buttons for playback control and various other functions. On the output side, they may also provide useful feedback through motor faders indicating the current values, LED strips showing meter values in real-time, a timecode display, and "scribble strips" (little LCD displays) to denote track and parameter information. The knobs and faders are typically organized into banks of 8 which can be switched at the push of a button to accommodate a large number of different parameters (which is why you need the scribble strips to figure out which tracks and parameters are actually represented on the control surface at any one time).

The first DAW controller was produced by the mixer manufacturer Mackie for Logic by emagic, and was subsequently modified to support a number of other DAW programs, see Fig. 1 [5]. The Mackie Control also set the de facto standard MIDI protocol for this kind of gear, although there are some alternatives, most notably the HUI protocol developed for Digidesign's Pro Tools.

Nowadays, DAW controllers can take many shapes and forms, ranging from tiny gadgets just providing playback con-

Figure 1: Mackie Control [5].

trols, keyboard controllers with added knobs and faders, and even foot controllers with switches and expression pedals, to full-blown mixer-like control surfaces. Most of these speak the Mackie Control Protocol (MCP), which is also supported by most DAW programs these days. Some prominent examples of these are the Mackie Control Universal, the Icon Platform M, Behringer's X-Touch series, as well as the Presonus Faderport controllers. There are also software implementations on mobile platforms, such as humatic's TouchDAW[4], which emulates a full Mackie-compatible DAW controller on Android devices, and can be connected to PCs either via USB or LAN (using RTP-MIDI or ipMIDI in the latter case); see Fig. 2.

MCP is in fact just a subset of MIDI, so it can be transmitted over any kind of MIDI connection, but it uses MIDI in its own, somewhat idiosyncratic way. Here is a brief summary of the most important features relevant for our purpose:[5]

- The knobs are usually rotary encoders which transmit relative changes in sign-bit encoding (thus, e.g., the CC values 1 and 65 denote an incremental change by +1 and -1, respectively). This includes pan (mapped to CC16 to CC23), and often there's also a big jog wheel (CC60) used to change the position of the playback cursor on the timeline.
- The faders emit pitch bend messages on the first eight MIDI channels (rather than MIDI CC) to take advantage of the 14 bit resolution these messages provide.
- The buttons used to control playback and other functions emit note messages such as note 94 and 93 for transport control start and stop. Thus MCP *always* needs a separate MIDI connection to the DAW where *only* MCP data is transmitted, lest you risk the knobs being pushed triggering actual notes in some synthesizer plugin.
- MCP controllers also *receive* data to properly set the current values of encoders and faders. In addition, on the back connection, channel pressure (monophonic aftertouch) messages are employed to denote meter values, MIDI CCs 66 to 73 represent the timecode display, and sysex messages encode the contents of the scribble strips.

[4] `https://www.humatic.de/htools/touchdaw/`

[5] Although MCP is widely used, there doesn't seem to be an official specification of the protocol anywhere on the internet. However, a fairly comprehensive overview of the protocol (albeit without the feedback messages) can be found at `http://www.jjlee.com/qlab/MackieControlMIDIMap.pdf`.

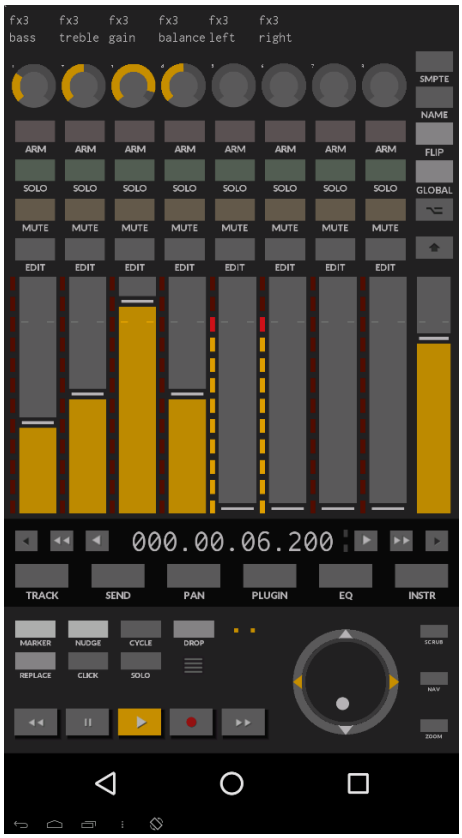

Figure 2: TouchDAW running on Android.

It is also worth noting here that while the encoders, faders, and transport controls should work the same with any DAW, the other (button) controls are much less standardized and may vary a lot in function depending on the DAW program that you use. Therefore many Mackie-compatible controllers ship with overlays for popular DAWs. Likewise, TouchDAW lets you configure the target DAW and changes some of its button layout and labeling accordingly.

3. THE FAUST-MCP PACKAGE

faust-mcp is distributed as open-source software on Github.[6] The package contains a Pd abstraction mcp.pd, along with some helper abstractions and externals, and a few examples. To use it, you'll obviously need Pd to run the patches, an installation of Grame's Faust compiler (and gcc) to compile your

[6] `https://github.com/agraef/faust-mcp`

Faust programs, and an MCP-compatible controller. We have tested the package with the Behringer X-Touch controllers (including the X-Touch One and Mini), the Studiologic Mixface, the Korg nanoKontrol2, and humatic's TouchDAW, but any MCP-compatible controller should work according to the capabilities it offers.

faust-mcp is built on top of pd-faust, and the accompanying externals are written in the author's Pure programming language [6], so both pd-faust and pd-pure need to be installed and enabled in Pd. Sources and binary packages for all of these can be found on the Pure website, which also provides detailed installation instructions.[7]

4. HOW IT WORKS

Basically, faust-mcp is a specialized MIDI mapper which translates MCP to standard MIDI control change (CC) messages and vice versa. Apart from the requisite MIDI bindings in the Faust programs, no manual setup is required; once the patch has been loaded, the mcp.pd abstraction keeps track of all the MIDI controls in all Faust dsps and configures itself accordingly in a fully automatic fashion. Note that in the current implementation, *only* controls with MIDI bindings will show on the MCP surface.

The faders and encoders of the MCP device are linked to the MIDI controls of your Faust dsps, so moving them changes the controls of the dsp accordingly. Conversely, changing the controls in the Pd GUI sets the controls of the device (if it supports feedback). In faust-mcp, the faders and encoders in each strip can be used interchangeably (and will move in lockstep if the device supports feedback), to accommodate any kind of MCP device which has any faders or encoders at all. *Passive* Faust controls (bargraph elements which output control values rather than reading them) are also supported and will be displayed using the meter strips of the MCP device if it has those (for instance, you can see these in the `left` and `right` strips of the `fx3` unit in Fig. 2).

The controls are organized into banks of eight faders and encoders. The abstraction provides as many banks as needed to represent all MIDI controls of all Faust dsps, ordering the controls by increasing MIDI CC numbers. The usual bank and channel controls on the MCP device can be used to switch between different banks as needed, so that all controls with MIDI bindings become accessible.

Scribble strips are also supported (as can be seen at the top of Fig. 2); they will show the name of the Faust units and controls assigned to each fader and encoder, or display the corresponding parameter values. Also, if you're using the included midiosc.pd abstraction, the transport keys of the device can be used to control playback. There are a number of other useful features like these, which will be described in Section 6.

The faust-mcp package contains a few examples which can be run straight from the source directory. The sources also include a small collection of sample Faust instruments and effects in the dsp subdirectory. Before running any of the examples, you'll have to compile these with the Faust compiler.

[7] See `https://agraef.github.io/pure-lang/`. Binary packages for Arch, Debian, and Ubuntu can be found on the Open Build Service, please check the "Pure on Arch" and "Pure on Debian/Ubuntu" wiki links on the website.

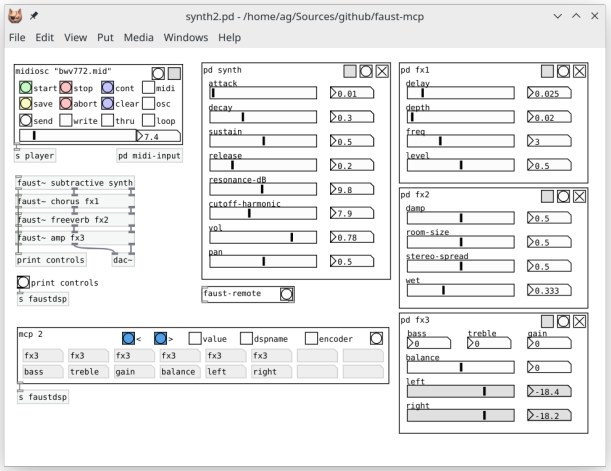

Figure 3: Pd patch running faust-mcp.

A Makefile is included, so you can just type `make` in the dsp folder to do this. The provided examples have all been set up so that the MCP device is expected to be connected to Pd's second MIDI port, so you'll have to configure your Pd MIDI connections accordingly. The included README file describes this in more detail.

Of course, you can also use the abstraction in your own patches. To do this, it's enough to copy the mcp folder to the directory containing your patch and Faust modules, or to any folder on Pd's library search path. To insert an instance of the abstraction into your patch, create an object (`Ctrl+1`) and type `mcp` followed by the MIDI port number to which the MCP device is connected.[8] Then connect the abstraction's single outlet to whatever `faust~` objects you wish to control, or just send it to the `faustdsp` receiver which is read by all Faust modules present in the patch. In either case, MIDI CC data emitted by the abstraction is encoded in the author's SMMF Pd message format[9], which is also the format used by pd-faust to encode all MIDI messages.

For instance, `mcp 2` connects to the device on Pd's second MIDI port. In principle any of Pd's MIDI ports can be used there (port 1 being the default). But as we already mentioned, MCP uses note and control data in its own peculiar way, thus you should make sure that live MIDI input to the Faust dsps is kept separate from the MCP data.

5. EXAMPLE

Fig. 3 shows the synth2.pd example from the faust-mcp package; please also revisit Fig. 2 to see how the same patch looks on the MCP device (TouchDAW in this case). In both figures the third (and last) bank of controls is shown. This example illustrates all the various elements: several `faust~` objects along

[8] Note that only a *single* instance of the mcp patch is needed for any running Pd instance, not one per toplevel patch!

[9] `https://bitbucket.org/agraef/pd-smmf`

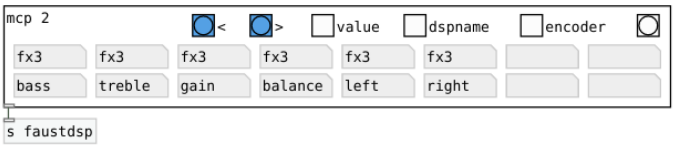

Figure 4: mcp abstraction closeup.

with their Pd GUIs, the `midiosc` abstraction which can be used to play back a MIDI file and record automation data, and the `mcp` abstraction itself. Note that faust-mcp ships with a special version of the midiosc abstraction which has been modified so that the MCP transport controls can be used with it.

Let's have a closer look at the `mcp` abstraction in the example (cf. Fig. 4). It shows a mirror of the scribble strips, as they will render on the MCP device, as well as a few buttons and toggles in the top row. All these functions are also available using corresponding controls on the MCP device, as described below; in the following list we give the equivalent MCP functions in parentheses.

- The first two bang controls, labeled < and >, switch to the previous and next bank of eight faders, respectively. (MCP: bank left/right keys)
- The `value` toggle, when engaged, shows the current values of the controls in the top row of the scribble strips. (MCP: touch a fader, or push an encoder)
- The `dspname` toggle switches the scribble strips between showing the instance and the dsp name of the Faust unit. (MCP: F1 key)
- The `encoder` toggle switches between two alternative display styles (`fan` and `pan`) for the encoder LED rings. Fan style (the default) shows an arc from 0 to the current value, while pan style shows just a single tick between min and max markers. (MCP: F2 key)
- The bang control on the right resets the internal state of the abstraction and re-displays the scribble strips. (MCP: F3 key)

Note that the controls in the abstraction are not meant to replace a real MCP device; they merely provide you with the most essential functions in case your MCP device lacks some of these controls. Also, the facsimile of the scribble strips will be helpful if your device has no display.

In the following section, we discuss the meaning of all available MCP controls in some detail.

6. CONTROLS

The primary purpose of the mcp abstraction is to take controller input from the mixer strips (faders and encoders) of your device and map them to the corresponding MIDI control changes of the Faust units in your patch. It also does the reverse translation, providing feedback to the MCP device (moving motor faders or lighting up LEDs if your device has any of those) if you change the Faust controls in the patch. In addition, the abstraction offers various other useful functions, mostly accessible through special keys on the MCP device:

- **Bank changes:** As already mentioned, the controls are organized into banks of size eight (which matches the number of strips on most MCP devices). The bank left and right buttons can be used to switch between these, so that all Faust controls become accessible. The channel left and right buttons, if available, move through the controls one strip at a time; this is useful, in particular, with single-strip devices like the X-Touch One.
- **Scribble strips:** Instance/dsp and control names are shown in the scribble strips of the device (if available), and touching the faders or pushing the encoders toggles the value display in the top line of each scribble strip.
- **Special dsp controls:** Each Faust dsp has three special controls, which correspond to the buttons in the upper right corner of the generated Pd GUI (cf. Fig. 3): `record` (a toggle which arms the unit for recording of OSC automation data when used with the midiosc abstraction), `reset` (a bang control which resets all controls to their initial values), and `active` (a toggle which turns the unit on or off). With the mcp abstraction these are assigned to the record, solo/select and mute buttons of the device, respectively. The rec and mute buttons also provide feedback, i.e., the buttons light up when the option is engaged. In the case of `mute` this actually means that the unit is *de*activated, so the corresponding GUI toggle is *off*. Pressing the select or solo button simply resets all controls of the dsp to their initial values, without lighting any buttons.

  Note that the special dsp controls always apply to the dsp *as a whole*, so pressing the button on *any* strip currently assigned to a given dsp will change the mute or record status of *all* the other buttons currently assigned to the same dsp.
- **Display options:** The following options are assigned to some of the function keys of the MCP device: F1 switches the scribble strips between instance and dsp name of the Faust units; F2 switches the encoder style between `fan` and `pan`, as discussed in the previous section; and F3 tells the abstraction to update its internal state and re-display the scribble strips (which can be used to force an update of the display, e.g., after editing and reloading Faust units).
- **Playback and transport:** When used with the included (modified) version of the pd-faust midiosc player, the transport controls will work as follows: the rewind key moves the playhead to the beginning of the MIDI file, fast forward moves it to the end; stop stops, and play toggles playback; record toggles the player's OSC automation recording; cycle toggles the player's loop function; and the big jog wheel and the cursor left/right keys move the playhead in smaller and larger steps, respectively. In addition, the function keys F4, F5 and F6 are assigned to some special OSC recording functions (`save`: save the currently recorded automation data to a text file; `abort`: delete the automation data of the current take; and `clear`: delete the entire automation sequence). Please check the pd-faust documentation for more details on how these operations are used.[10]

[10] `https://agraef.github.io/pure-docs/pd-faust.html`

- **Timecode:** When used with the midiosc player, the timecode display shows the time (in h/m/s/tenths of seconds) of the current playhead position.

Obviously, some of these functions may or may not be available depending on the MCP device that you have. The Mackie, Faderport 8 and X-Touch devices should enable all features, but some lesser MCP devices may not offer transport or function keys, push encoders, fader touch detection, scribble strips, or a timecode display.

Finally, let us mention in passing that even if your MIDI controller does *not* have built-in MCP support, chances are that if it has enough faders, knobs and buttons, you can make it work as an MCP-compatible device using the author's midizap program [7]. For instance, faust-mcp works just fine with the Akai APCmini, or even the Harley Benton MP-100 foot controller, using the corresponding MCP emulations included in the midizap distribution.

7. FUTURE WORK

While faust-mcp is perfectly usable already, we still consider it work in progress. Here are some things we may want to address in future versions:

- The most notable limitation right now is that faust-mcp only covers dsp controls which already have MIDI bindings. This simplifies the implementation a lot. However, another option would be to go through pd-faust's OSC layer instead. This would allow arbitrary controls to be mapped, without having to configure MIDI bindings beforehand.
- It would be nice to offer more layout options (i.e., how "pages" for different Faust units are organized, and how the controls are ordered).
- Currently controls mapped to the same MIDI CC in different Faust units will be mapped to the same MCP control. This is an outright bug and will hopefully be fixed by the time you read this.
- faust-mcp is currently hard-wired to use 8-fader banks, which is what most dedicated DAW controllers offer. But there are devices with smaller and larger bank sizes (as well as extender units which can be added to existing DAW controllers), so it makes sense to provide alternative versions of the mcp abstraction to accommodate all common sizes.
- For DAW controllers without motorized faders, the current fader positions will often be way off from the actual Faust control values, especially after bank switches. The usual way to deal with this is a "pickup" (a.k.a. "takeover") mode which makes sure that controls start moving only when the fader "picks up" the actual value. Obviously, it would be nice to have this in faust-mcp as well, at least as an option.[11]
- There should be some form of musical timecode display. Currently only physical time in h/m/s/tenths is shown. This is due to limitations in the current pd-faust implementation which doesn't report musical time.

[11] As a remedy for the time being, if your controller doesn't have motor faders, then it may be safer to just use the encoders of your device instead, because these always emit changes relative to the current control value.

8. REFERENCES

[1] Yann Orlarey, Albert Gräf, and Stefan Kersten, "DSP programming with Faust," in *Proceedings of the 4th International Linux Audio Conference*, Karlsruhe, 2006, pp. 39–47, ZKM.

[2] Albert Gräf, "Pd-Faust: An integrated environment for running Faust objects in Pd," in *Proceedings of the 10th International Linux Audio Conference*, Stanford University, California, US, 2012, pp. 101–109, CCRMA.

[3] Yann Orlarey, Dominique Fober, and Stephane Letz, "Syntactical and semantical aspects of Faust," *Soft Computing*, vol. 8, no. 9, pp. 623–632, 2004.

[4] Colby N. Leider, *Digital Audio Workstation*, McGraw-Hill, Inc., New York, NY, USA, 2004.

[5] Mark Wherry, "Mackie control : DAW control surface," Sound On Sound, Dec. 2003, `https://www.soundonsound.com/reviews/mackie-control-universal`. Last access: Dec. 2019.

[6] Albert Gräf, "Signal processing in the Pure programming language," in *Proceedings of the 7th International Linux Audio Conference*, Parma, 2009, Casa della Musica.

[7] Albert Gräf, "midizap: Controlling multimedia applications with MIDI," in *Proceedings of the 17th International Linux Audio Conference*, Stanford University, California, US, 2019, pp. 113–120, CCRMA.