NONLINEARLABS

Content

1 Current Situation (legacy LPC code)

The LPC code did NOT use any settings made in the User Interface at all, the UI was a complete dummy, the four selectable settings were ignored. Rather, the LPC was fixed at using the „pot, tip active“ setting for all four pedals. Pedals would work correctly only when plugged in at boot time, and the user had to cycle through the full pedal travel to establish a correct auto-range setting every time, after booting.

2 Current Situation (new LPC code)

The new LPC code now handles the submitted settings from the UI correctly. Internally, the four settings („pot, tip active“, „pot, ring active“, „switch, closing“ and „switch, opening“), given as a number 0...3 in the corresponding BB messages, are mapped to hard-coded presets which contain the parameters to the actual setup calls of the EHC API.

The pedal settings, among other global device settings are submitted by the UI every time it sends the edit buffer. This is a bit sketchy and doesn't make sense (global settings are NOT associated to presets and thus don't need to be transmitted on every preset change or load), but it is the way it is, chosen for historic reasons of convenience.

This behavior required that the LPC detects if the same selection for a pedal is send again and then does NOT do a full reset of the pedal settings, notably it doesn't reset the auto-ranging (see „ExternalHardwareControllers_API_V1.0 “ document for details). But the LPC now also keeps the auto-range internal data across reboots (stored in EEPROM) which leads to the following problem: The auto-ranging can only ever be reset either by unplugging and re-plugging the pedal while the C15 is on, or by selecting a different „pedal type“ (note that „switch, closing“ and „switch, opening“ is not a different type internally, as only the output values are inverted, but no change of the auto-ranging is made as it has nothing to do with that inversion).

3 New implementation (with new LPC code)

3.1 UI appearance

There is little change in the UI appearance, at least for hardware-UI. The main difference is that the list of pedal types is vastly extended to some 20+ entries. Some of the entries are specific for a given make and model of pedal, for user convenience. Other entries reflect the generic settings as before, and completely new types have been added („adjustable resistors“ and „control voltage“).

One specific detail changes, though, that is the actual selection of a new type. Formerly, this was similar to a preset change with „Direct Load“ set to ON, stepping through the list had immediate effect. This will change to a behavior similar to „Direct Load“ OFF: The user can freely navigate in the list, with the currently effective preset shown inverted while the selected preset gets a wire frame. The user has to explicitly acknowledge the selection, and then (and only then) the pedal is reset, regardless if it has been the previously effective selection or not. By this, the user can either exit the selection screen without any effect, or acknowledge any change of selection (including none), thus resetting the pedal.

From the main pedal screen (which remains as is in appearance and function), we have the following changes for the actions:

- Softkey 3/4 or „Edit“ or Encoder Button go to to pedal setup screen.
- „Enter“ does nothing, there is nothing to confirm here (I understand „Enter“ was misused to enter the next screen at many points in the UI, but the official navigation buttons are the soft-keys)

In the pedal setup screen:

- „Dec“, „Inc“ and Encoder navigate through the pedal preset list.
- „Enter“ or Encoder Button make the selected preset the active one, reset the pedal, but they do NOT exit the screen.
- Soft key 2 or „Edit“ quit the screen without effect and return to the previous.

By this, the user can quickly reset the current pedal by pressing the encoder button twice. The UI logic is very similar to that used for the presets in „Direct Load“ OFF mode which should be quite intuitive to the user. Note this is a preliminary suggestion, perhaps SSC wants to have the last word on it.

For the Web-UI, the changes would be similar. Again, a changed selection must be explicitly confirmed and the pedal type is transmitted only on that confirmation, regardless if the selection is actually a different one or not (thus affording future pedal reset logic).

These changes in UI control flow could be easily implemented and tested first with no other change in the logic behind it, that is, not changing the internal actions when a new setting is actually sent down to the LPC, it could remain the current „set pedal type“ messages, using the four types of pedals available.

3.2 Message implementation

In contrast to the current implementation where the pedal type is transmitted as a simple number and the LPC makes sense of that itself and calls the EHC setup/parametrization with corresponding hard-coded data, the new message implementation will need to load and send all the setup data for an EHC controller explicitly, using various commands from the EHC API. The legacy calls shall not be used anymore, even they are still implemented in LPC.

The general procedure when confirming a pedal preset in the UI is to consecutively send a number of EHC messages:

- disable controller processing
- delete any controllers attached to the current input jack (tip and ring). This affords the reset of auto-ranging, and removes any controller still active at the adjacent contact of the selected input (a scenario that can happen when 2-wire controllers are used).
- send main configuration (contents of EHC_CR configuration register)
- send dead-zones for auto-ranging (when present in the preset)
- send manual range limits (when present in the preset)
- enable controller processing

Whereas, when sending the global settings upon initial edit buffer and preset change/load:

- disable controller processing
- for (all pedals)
  - if the „pot“-field in the main configuration EHC_CR is false, delete adjacent controller (for the same reason as above).
  - send main configuration (contents of EHC_CR configuration register)
  - send dead-zones for auto-ranging (when present in the preset)
  - send manual range upper and lower limits (when present in the preset)
- enable controller processing

The individual messages for the pedals require that the EHC controller number (CTRLID), plus hardware source (HWSID) for the configuration register call, have to be set up properly in the transmitted data.

The controller number is directly corresponding to the hardware input used, so it must be known before if the pedal hardware is ring active or tip active. This can be implemented by setting bit 0 of CTRLID right in the preset data, and then ORing in („Pedal Number“ * 2) for the actual calls, see section „7 Compatibility“ in the EHC API. The HWSID directly corresponds to the „Pedal Number“ which is the equivalent of the TRS jack number it is attached to, for this first EHC implementation.

3.2.1 Possible Implementation in the Playground

Currently, the pedal setting are stored in „settings.xml“ with four entries:

```
<setting id="Pedal1Type">pot-tip-active</setting>
<setting id="Pedal2Type">pot-tip-active</setting>
<setting id="Pedal3Type">switch-opening</setting>
<setting id="Pedal4Type">pot-tip-active</setting>
```

This was already all the required data. With the new pedal functions the list of possible entries would vastly expand. Rather than hard-coding what the ID strings are and what is associated with them, I suggest using an extra file for the EHC presets (with a hard-coded fallback if the file is missing or corrupt, generating a new file). This would allow updating of the available EHC presets on-the-fly when we would like to add more pedal presets, no need to do a full firmware update for that, and it would allow editing/creating pedal presets in the future (MS1.8++), together with a full-blown usage of the EHC's possibilities (like 8 controllers, arbitrary assignment of HWSID, etc)

The pedal preset file could basically look like this (though most possibly it would be .xml style, not classic .ini style):

```
[presetlist]
"--Expression--"
pot-ring-active
fractal-ev-2
"--Damper--"
korg-ds-1h
„----“
off

[pot-ring-active]
displayname="Pot Ring Active"
hswid=auto
ctrld=auto,ringactive
continuous=true
silent=false
pot=true
pullup=false
invert=false
autoranging=true
autoholdstrength=2
deadzones=
rangelimits=
info="Generic Expression Pedal, Ring Active (YAMAHA-style connector pinout).\n"
"If you feel the progression of values vs. travel doesn't feel linear, please try a \"Tip
Active\" setting.\n"
"If your pedal has a \"min\" adjust, please set it fully counter-clockwise."

[fractal-ev-2]
displayname="Fractal EV-2"
hswid=auto
ctrld=auto,tipactive
continuous=true
silent=false
pot=true
pullup=false
invert=false
autoranging=true
autoholdstrength=1
deadzones=1,1
rangelimits=

[korg-ds-1h]
displayname="Korg DS-1H"
hswid=auto
ctrld=auto,tipactive
```

```
continuous=true
silent=false
pot=false
pullup=true
invert=false
autoranging=true
autoholdstrength=2
deadzones=
rangelimits=

[off]
displayname="--off--"
hwsid=15
```

At first the layout of the scrolling list would be defined, how many entries there are and what is their ordering. SSC suggested that it might be useful to insert blank or comment lines (which are not selectable, of course). The use case would be to allow for some logical grouping of entries.

Then, each preset would have its data section, with the following fields:

- displayname : must be given, enclosed in „“, not blank/whitespace only
- hwsid : either *auto* (default, corresponds to TRS hardware jack number 0..3), or a fixed HWSID name (ehc0..ehc8, pb, at, r1, r2, off). Currently, only HWSID=off makes sense, indication a disabled („off“) controller.
- ctrlid : either *auto,tipactive* (default) or *auto,ringactive* . Auto indicates that the CTRLID shall be set according to the TRS hardware jack number (see above), using the additional +1 offset when ringactive is given.
- continuous : either *false* (default) or *true*. Determines whether the controller shall work in continuous or bi-stable mode.
- silent : either *false* (default) or *true*. When true, the controller does NOT send data to the Audio Engine.
- pot : either *false* (default) or *true.* Indicates whether the controller is a 3-wire element (potentiometer,) or a 2-wire element (resistor, switch, control voltage)
- pullup : either *false* (default) or *true*. Indicates whether the controller needs a pullup on the main ADC channel or not. Must be set to false for pots and control voltages, and set to true for resistors and switches.
- invert : either *false* (default) or *true*. When true, the final output from the controller is inverted.
- autoranging : either *false* (default) or *true*. When true, auto-ranging is enabled.
- autoholdstrength : integer value, either none (defaults to 2) or a number between 0(=off) and 4(extreme)
- deadzones : two integer values, comma separated, either none (defaults to dead zone data not transmitted), or two values between 0..20, setting the lower and upper dead zone percentages. Will have no effect if autoranging=false
- rangelimits : two integer values, comma separated, either none (defaults to manual rangelimits data not transmitted), or two values between 0..65535. Will have no effect if autoranging=true.
- an additional field for context-sensitive help text (displayed with Info button) might be a good idea.

At the moment, the presets will be:

```
"Pot Tip Active"            (3-wire continuous, tip active)
"Pot Ring Active"           (3-wire continuous, ring active)

"Pot Tip Act. Rev."         (3-wire continuous, tip active, reverse action)
```

```
"Pot Ring Act. Rev."        (3-wire continuous, ring active, reverse action)

"Resistor"                  (2-wire continuous, on tip)
"Resistor Rev."             (2-wire continuous, on tip, reverse action)

"Switch Closing"            (2-wire bi-stable, on tip)
"Switch Opening"            (2-wire bi-stable, on tip, inverted)

"CV 0..5 V"                 (2-wire continuous, on tip, fixed range)
"CV Auto-Range"             (2-wire continuous, on tip, fixed range)

"Roland EV-5"               (3-wire continuous, tip active, with "min")
"Yamaha FC7"                (3-wire continuous, ring active)
"Boss FV500L"               (3-wire continuous, tip active, with "min")
"Moog EP-3 (std)"           (3-wire continuous, tip active, with "range")
"Doepfer FP5"               (3-wire continuous, ring active)
"Fractal EV-2"              (3-wire continuous, tip active)
"Lead Foot LFX-1"           (3-wire continuous, tip active)

"Roland DP-10 (cont.)"      (3-wire continuous, tip active)
"Korg DS-1H"                (2-wire continuous, on tip, reverse action)
"Yamaha FC3A"               (3-wire continuous, ring active, reverse action)

"Off"
```

The list will most probably increase to some more known types before release.

Once the exact format for the pedal preset file is known, I will provide the detailed settings for each preset.

For the formats of the required LPC messages, see EHC API. For their implementation (and anything LPC related) I prepared a set of shared files under projects/shared:

„lpc-defs.h“ contains all the definitions of the LPC messages as enums, and „lpc-converters.h“ contains converters to read and write the EHC bit field implementation data for EHC_CR and EHC_SR (the latter will be used in future). I am already using these shared definitions in the LPC code and in the helper tools running on the BBB), to keep them in sync. Have a look at the code of „projects/realtime-system/supplements/native-bbbb/“ for examples.

At the moment, the definitions are declared for usage in plain C but maybe this can be transparently converted so that the same files can be used both with my C code and the C++ that is used for playground etc, using all the bells and whistles (classes with methods, namespaces etc), via `#ifdef __cplusplus` splitting.

Currently, all this is only available in the C15/LPC branch but it will hopefully be merged soon into the master branch, so that all the playground code (and audio engine as well) may use the same shared definitions in future (I'm aware that means a rewrite of projects/playground/src/proxies/lpc/LPCProxy.h etc)