Developing Bass Guitar Effects

for the Cleveland Music Co. Hothouse DSP Pedal

A Complete Guide to Pure Data Patch Development, Compilation, and Deployment

Introduction

The Cleveland Music Co. Hothouse is a programmable guitar/bass effects pedal built around the Electro-Smith Daisy Seed DSP platform. It provides musicians with a powerful, customizable effects unit that can run patches created in Pure Data using the PlugData visual programming environment.

This guide covers the complete workflow for developing bass guitar effects, from initial patch design through compilation and deployment to the hardware.

Development Chain Overview

The development workflow follows a streamlined path from visual programming to deployed firmware:

```
PlugData (visual patch) → HVCC Compiler → C++ Code → Daisy Seed Flash
```

1. Design your patch in PlugData with "Compiled Mode" enabled
2. Use only HVCC-compatible Pure Data objects
3. Map hardware controls via @hv_param annotations
4. Compile and flash directly from PlugData to the Hothouse

Hothouse Hardware Layout

The Hothouse pedal provides a comprehensive set of programmable controls optimized for guitar and bass effects:

Physical Components

Component	Quantity	Type	Notes
Potentiometers	6	Linear 10K	Values 0.0-1.0
Toggle Switches	3	3-position	ON-OFF-ON
Footswitches	2	Momentary SPST	Press detection
LEDs	2	Standard 3mm	PWM controlled
Audio Jacks	4	1/4" TRS	Stereo I/O

Control Mapping Reference

Control	Type	Receive Name	Range/Values
Knob 1-6	Potentiometer	knob1 to knob6	0.0-1.0
Toggle 1-3	3-position switch	sw1 to sw3	0.0, 0.5, 1.0
LED 1-2	Output LED	led1, led2	0.0-1.0 (PWM)

Control	Type	Receive Name	Range/Values
Footswitch L/R	Momentary	bypass, cycle	0 or 1
Audio In	Stereo	adc~ 1 2	L/R channels
Audio Out	Stereo	dac~ 1 2	L/R channels

Parameter Annotation Syntax

Hardware controls are mapped using the @hv_param annotation on receive objects. This tells the HVCC compiler to expose the parameter to the hardware interface.

Basic Syntax

```
r <name> @hv_param [min] [max] [default]
```

Examples

```
r knob1 @hv_param 0 1 0.5       // Standard knob, 0-1 range, default 0.5
r mix @hv_param 0 100 50        // Mix percentage 0-100%
r freq @hv_param 20 20000 1000  // Frequency in Hz
r feedback @hv_param 0 0.9 0.4  // Feedback with safe maximum
```

Essential DSP Patterns for Bass Effects

Parameter Smoothing

Always smooth control inputs to prevent audio clicks and zipper noise:

```
[r knob1 @hv_param]
    |
[pack f 20]          // value, ramp time in ms
    |
[line~]              // audio-rate smoothing
```

Dry/Wet Mix Control

Essential for most bass effects to preserve low-end clarity:

```
[inlet~]
    |
[s~ dry]

// ... effect processing ...

[catch~ effect_out]
    |
[*~]<-----------[r knob6 @hv_param]  // mix knob
    |                    |
    |              [pack f 20]
    |                    |
    |               [line~]
[+~]<---[r~ dry]
    |      |
    |   [*~]<--[- 1]      // (1 - mix) for dry signal
    |
[outlet~]
```

Bypass Switching with LED Indicator

```
[r bypass @hv_param]
    |
[change]
```

```
    |
[sel 1]                // trigger on press only
    |
[t b b]
    |
[f 1]                  // toggle flip-flop
    |
[== 0]
    |
[t f f]
 |   |
[s bypass_state] [s led1 @hv_param]  // LED reflects state
```

Bass-Specific Effect Templates

Bass Overdrive/Distortion

Knob1=gain, Knob2=tone, Knob3=blend (dry/wet), Knob4=level:

```
[adc~ 1]
    |
[s~ bass_dry]            // preserve clean signal

// Gain stage (1-50x)
[r knob1 @hv_param 0 1 0.5]
    |
[* 49] -> [+ 1] -> [pack f 10] -> [line~]
    |
[*~]                     // boost input
    |
[clip~ -0.7 0.7]         // soft clip
    |
[*~ 1.5] -> [clip~ -1 1]  // harder edge

// Tone filter (500-5000 Hz)
[r knob2 @hv_param] -> [* 4500] -> [+ 500] -> [lop~]

// Blend with dry (preserve low end)
[r knob3 @hv_param]      // blend control
[r~ bass_dry] [wet_signal]
    |              |
  [*~]<--[1-blend]  [*~]<--[blend]
    |              |
    +-----[+~]-------+

// Output level
[r knob4 @hv_param] -> [*~] -> [dac~ 1 2]
```

Bass Chorus

Knob1=rate, Knob2=depth, Knob3=mix:

```
[adc~ 1]
    |
[s~ chorus_in]

[delwrite~ ch_buf 100]

// Rate (0.1-2 Hz - slower for bass)
[r knob1 @hv_param] -> [* 1.9] -> [+ 0.1]
    |
[phasor~] -> [* 6.28318] -> [cos~]

// Depth (2-12ms modulation)
[r knob2 @hv_param] -> [* 10] -> [+ 2]
    |
[*~]                     // modulated depth
    |
[+ 20]                   // center delay 20ms
    |
[vd~ ch_buf]
```

```
    |
[*~ 0.7]                // wet level
    |
[+~]<---[r~ chorus_in]  // mix with dry
    |
[dac~ 1 2]
```

Bass Delay

Knob1=time, Knob2=feedback, Knob3=filter, Knob4=mix:

```
[adc~ 1]
    |
[throw~ del_input]

[catch~ del_input]
    |
[delwrite~ del_buf 2000]   // 2 second max

// Delay time (50-2000ms)
[r knob1 @hv_param] -> [* 1950] -> [+ 50]
    |
[pack f 20] -> [line~]
    |
[vd~ del_buf]
    |
// High-cut filter on feedback (removes harsh highs)
[r knob3 @hv_param] -> [* 4000] -> [+ 500] -> [lop~]
    |
// Feedback (0-0.85 - safe for bass frequencies)
[r knob2 @hv_param] -> [* 0.85]
    |
[*~] -> [clip~ -0.95 0.95] -> [throw~ del_input]

// Mix dry/wet
[r knob4 @hv_param]
[adc~ 1] [delayed_signal]
   |          |
 [*~]<-[1-mix] [*~]<-[mix]
   |          |
   +----[+~]---+
        |
   [dac~ 1 2]
```

HVCC Supported Pure Data Objects

The HVCC compiler only supports a subset of Pure Data objects. Using unsupported objects will cause compilation errors.

Signal Objects (Audio Rate)

Category	Objects
Math	*~ +~ -~ /~ abs~ clip~ max~ min~ pow~ sqrt~ wrap~
Oscillators	osc~ phasor~ noise~ cos~ tabosc4~
Filters	lop~ hip~ bp~ vcf~ biquad~ rpole~ rzero~ hilbert~
Delay	delwrite~ delread~ delread4~ vd~
Audio I/O	adc~ dac~ inlet~ outlet~
Utilities	sig~ line~ snapshot~ env~ samphold~ samplerate~
Communication	send~ receive~ s~ r~ throw~ catch~

Control Objects

Category	Objects
Math	!= % & && | || * + - / < << <= == > >= >> abs cos sin tan
Flow Control	bang change clip del delay float int line metro moses route sel spigot swap trigger unpack until
Communication	r receive s send print
Data	table tabread tabwrite
Structure	inlet outlet pd declare

Unsupported Objects (DO NOT USE)

These objects will cause compilation errors:

```
expr expr~ vline~ array soundfiler value text textfile list clone
netreceive netsend openpanel savepanel key keyname keyup stdout
```

Critical Development Rules

1. **Enable Compiled Mode** in PlugData before creating patches. This validates HVCC compatibility in real-time.
2. **Always smooth parameter changes** with line~ to prevent audio clicks and zipper noise.
3. **Use sig~ for control-to-signal conversion** - many signal objects require signal-rate inputs.
4. **No expr~ or expr** - use discrete math objects (*~, +~, clip~) instead.
5. **No soundfiler** - tables must be initialized differently for embedded systems.
6. **Test incrementally** - start simple, add complexity one feature at a time.

Compilation and Deployment Workflow

Compilation Steps

1. Open PlugData and navigate to Menu → Toggle "Compiled Mode"
2. Create your patch using only supported objects (watch for red outlines on invalid objects)
3. Go to Menu → Compile → Select "Electro-Smith Daisy"
4. Set Target Board to "Hothouse" (pre-configured in PlugData)
5. Choose Export Type: "Flash" for direct upload, or "Binary" for .bin file
6. Connect Hothouse via USB and click Export

Patch Size Settings

Size	Memory	Best For
small	512KB SRAM	Distortion, EQ, simple modulation effects
big	64MB SDRAM	Delays up to 2 seconds, reverbs, multi-tap delays
huge	Maximum	Loopers, very large reverbs, complex multi-effects

DFU Mode (Device Firmware Update)

If automatic flashing fails, manually enter DFU mode:

Hardware Method:

1. Hold BOOT button on Daisy Seed
2. Press and release RESET button
3. Release BOOT button - Daisy is now in DFU mode

Bootloader Shortcut: If your firmware includes CheckResetToBootloader(), hold the LEFT footswitch for 2+ seconds. LEDs will flash alternately 3 times, then enter DFU mode.

Debugging Tips

- Use the [print] object to send debug messages to the PlugData console
- Check console for HVCC compatibility warnings when Compiled Mode is enabled
- Red-outlined objects indicate unsupported objects - replace them
- Start simple - verify audio passthrough before adding effects
- Test each control mapping individually before combining

Common Issues and Solutions

Issue	Solution
No audio output	Verify adc~ and dac~ channel numbers (1, 2), check cable connections
Controls not responding	Verify @hv_param attribute on receive objects, check spelling

Issue	Solution
Compile errors	Enable Compiled Mode, remove unsupported objects, check arguments
Flash failures	Try different USB cable (data cable), install drivers, different port
Audio clicks/pops	Add parameter smoothing with line~, check for DC offset
Feedback runaway	Add clip~ objects to feedback paths, limit feedback < 0.95

Memory Considerations

Daisy Seed Memory Map

Region	Size	Speed	Use For
Internal SRAM	512KB	Fast	Code execution, small buffers
External SDRAM	64MB	Slower	Large delay lines, reverb buffers
QSPI Flash	8MB	Slowest	Program storage

Buffer Size Guidelines for Bass Effects

Effect Type	Recommended Max	Patch Size
Chorus/Flanger	50ms (2400 samples)	small
Short Delay	500ms (24000 samples)	small/big
Long Delay	2000ms (96000 samples)	big
Reverb	Variable	big
Looper	10+ seconds	huge

Quick Start Checklist

Before deploying your bass effect:

- PlugData is in Compiled Mode
- No red-outlined (unsupported) objects in patch
- All knob parameters use @hv_param annotation
- Parameter changes are smoothed with line~
- Audio I/O uses correct adc~/dac~ channels
- Feedback paths are limited to prevent runaway
- Patch size matches memory requirements
- USB data cable (not charge-only) is connected
- Daisy Seed is in DFU mode if needed

Happy effect building!