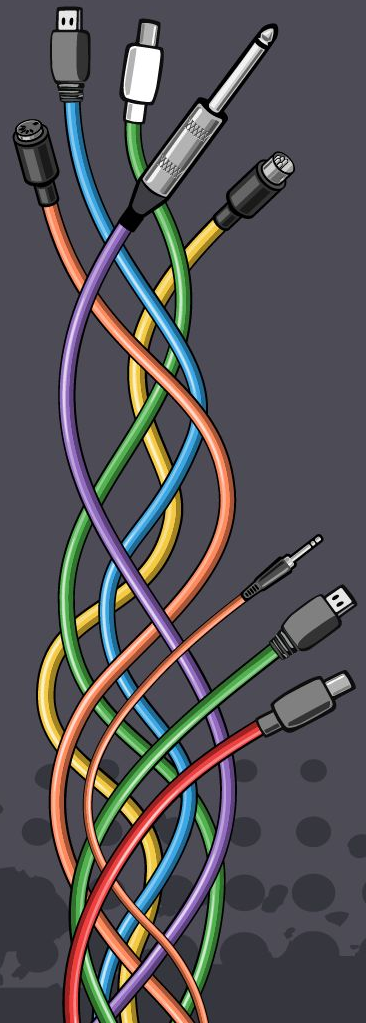


WORKSHOP:
DSP IN PRACTICE
FROM BLOCK DIAGRAM TO WORKING PLUGIN

JAN WILCZEK &
LINUS CORNELIUSSON

ADC25 Bristol

WolfSound DSPPro

- TheWolfSound.com
- youtube.com/@WolfSoundAudio
- WolfTalk podcast host
- JUCE YouTube channel creator
- ADC Mentor

Jan Wilczek
[Yan Vil-check]

JUCE

BOGREN DIGITAL

Linus Corneliusson

- BogrenDigital.com
- FascinationStreet.se
- Former mixing/editing engineer
- Ihsahn, James LaBrie, many more…
- Now: Plugin/DSP developer

slido

Is it your first time at ADC?

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

Outline

1. Getting the code to run
2. Plugin from scratch: Part I
    a. with DSP research
3. Plugin from scratch: Part II
    a. with the Python prototype
4. Surprise 1
5. **Break at 3 PM**
6. Plugin from scratch: Part III
    a. with C++/JUCE implementation
7. Summary
8. Surprise 2
9. Surprise 3

"Create a flanger as a VST3 plugin.

You have 24 hours."

Demo

What we'll create today

Task 0: Make the code run

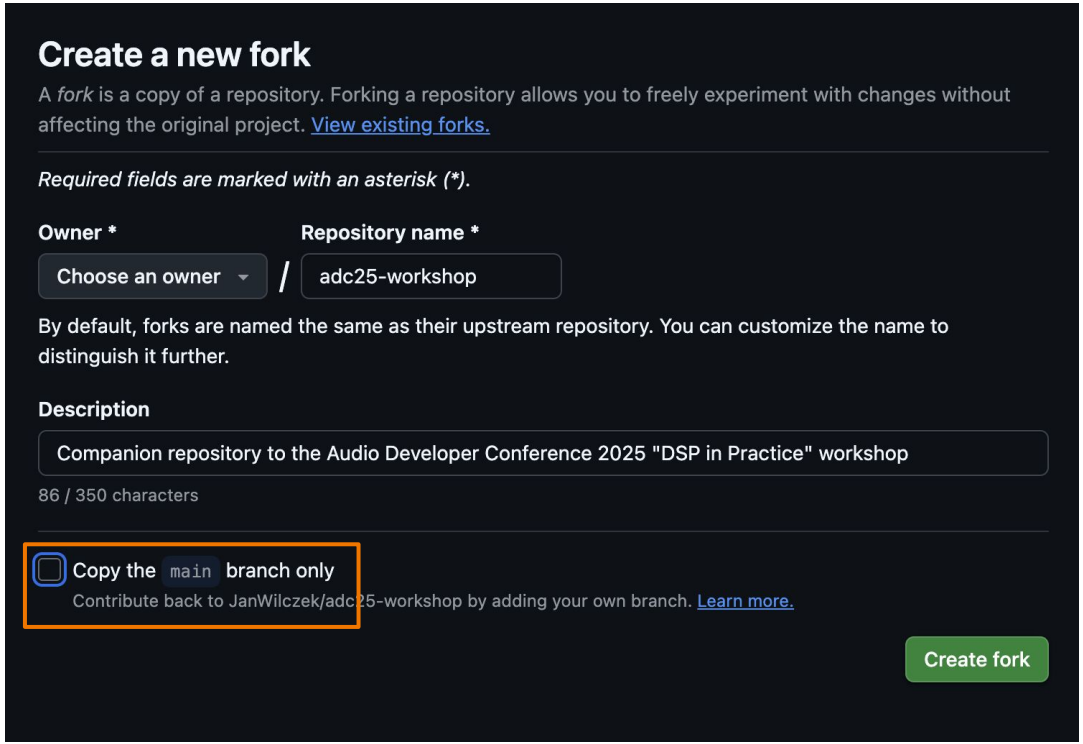
Repo: github.com/JanWilczek/adc25-workshop

Goal: Execute these two commands:

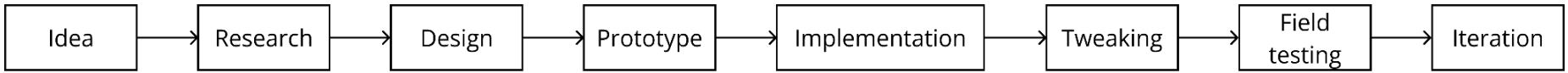
- `python py/main.py data/guitar_5th.wav`
- `cmake --build --preset default`

➡️ *"Getting Started"* section of the README file

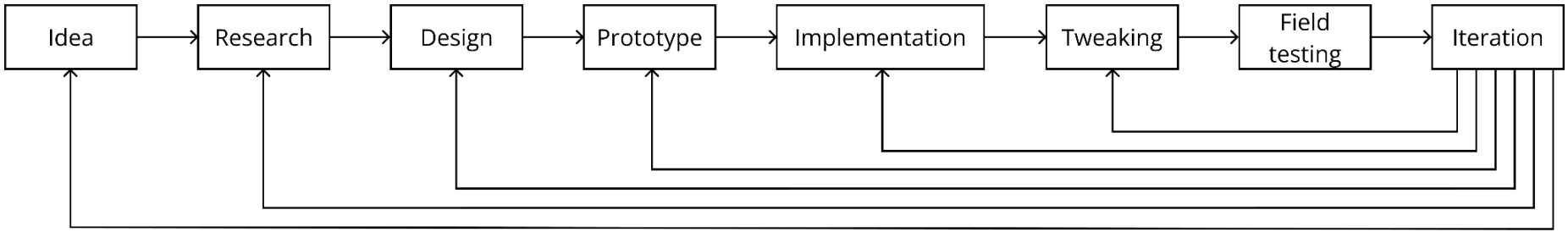
GitHub Codespaces: Fork the repo with all branches



Overview of the plugin creation process



Overview of the plugin creation process



Idea

- What do you want to create?
- Why? How is it useful?
- For whom is it useful?
- Can I explain the above in plain words?

"Create a flanger as a VST3 plugin.

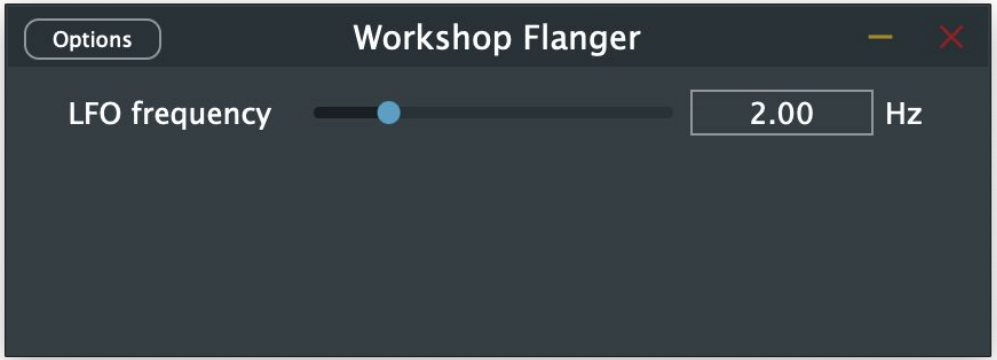
You have 24 hours."

Idea

Plugin we'll implement today: Flanger

Input:

Output:



Research

- Has anyone done it before?
- Is there a baseline
  (a plugin or a piece of code that does a similar thing)?
- How to do it (better)?
- End goal: a design

Research sources

- Books
- Research papers
- Online blogs & videos
- Online forums
- External consultancy

Examples of good resources

- Books
  - *Digital Audio FX* by Udo Zölzer et. al.
  - *Designing Audio Effect Plugins In C++* by Will Pirkle
- Papers
  - DAFX conference
  - Audio Engineering Society library
  - Google Scholar
- Online blogs & videos
  - WolfSound 🐺

Examples of good resources

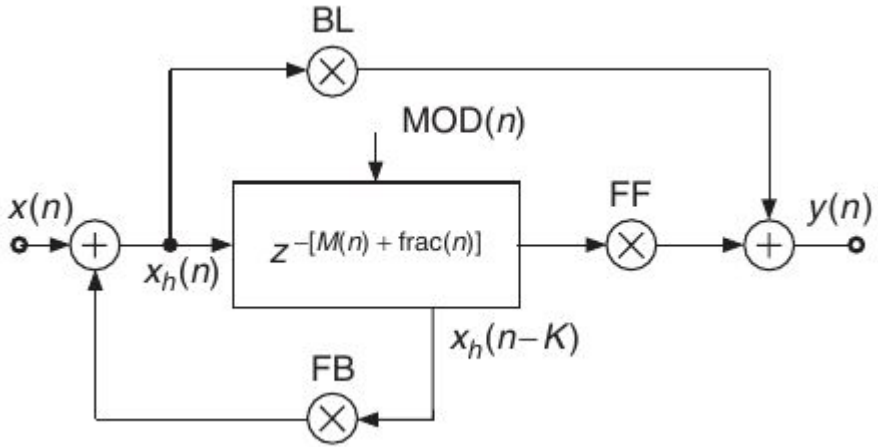
- Forums
  - JUCE forum
  - KVR Audio developer forum
  - DSP subreddit
  - DSP Stack Exchange
  - DIYStompBoxes
  - lines forum

Design

- DSP diagram
- Difference equations
- Textual description
- Literature reference
  - ”We will implement equations X-Y from paper Z”

Literature review

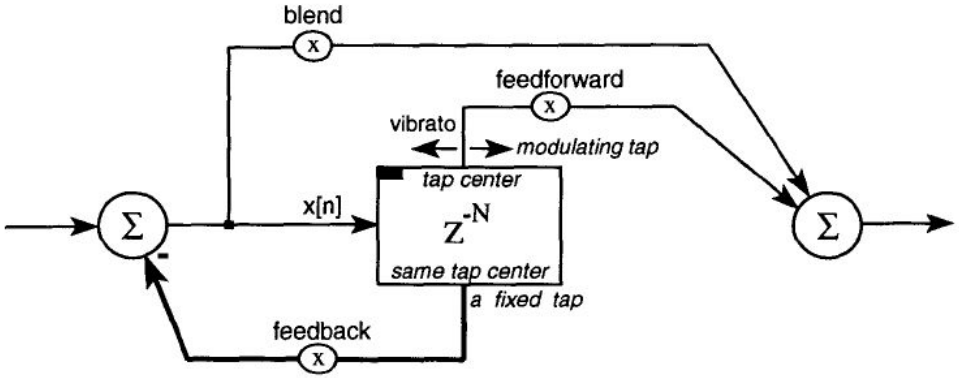
Udo Zölzer et. al.
Digital Audio FX
2nd ed. (2011)



	BL	FF	FB	DELAY	DEPTH	MOD
Vibrato	0	1	0	0 ms	0–3 ms	0.1–5 Hz sine
Flanger	0.7	0.7	0.7	0 ms	0–2 ms	0.1–1 Hz sine
(White) Chorus	0.7	1	(−0.7)	1–30 ms	1–30 ms	Lowpass noise
Doubling	0.7	0.7	0	10–100 ms	1–100 ms	Lowpass noise

Literature review

J. Dattoro. Effect design, part 2: *Delay-line modulation and chorus*. J. Audio Eng. Soc., 45(10): 764–788, October 1997.

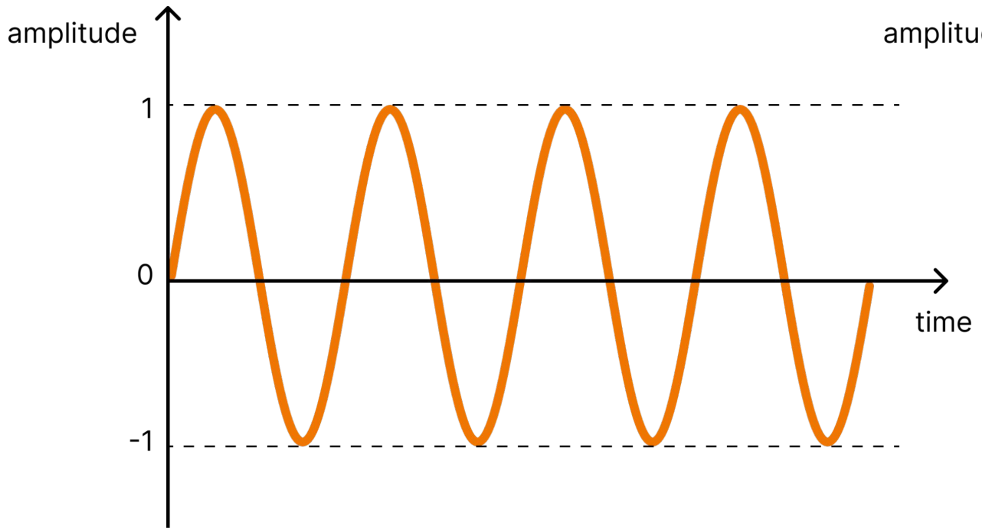


Effect	Blend	Feedforward	Feedback
Vibrato	0.0	1.0	0.0
Flanger	0.7071	0.7071	−0 7071
Industry standard chorus	1.0	0.7071	0.0
White chorus	0.7071	1.0	0.7071
Doubling	0.7071	0.7071	0.0
Echo[82]	1.0	≤1.0	<1.0

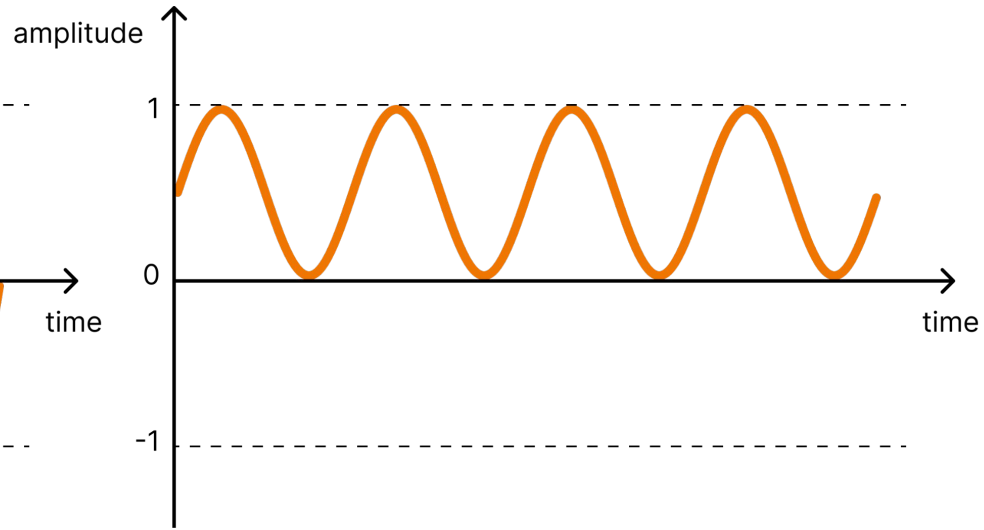
Table 7. Approximate effect delay range in milliseconds.

Effect	Onset	Nominal	Range End
Vibrato[83]	0	Minimal	5
Flange	0	1	10
Chorus	1	5	30
Doubling	10	20	100
Echo	50	80	∞

LFO: bipolar vs unipolar

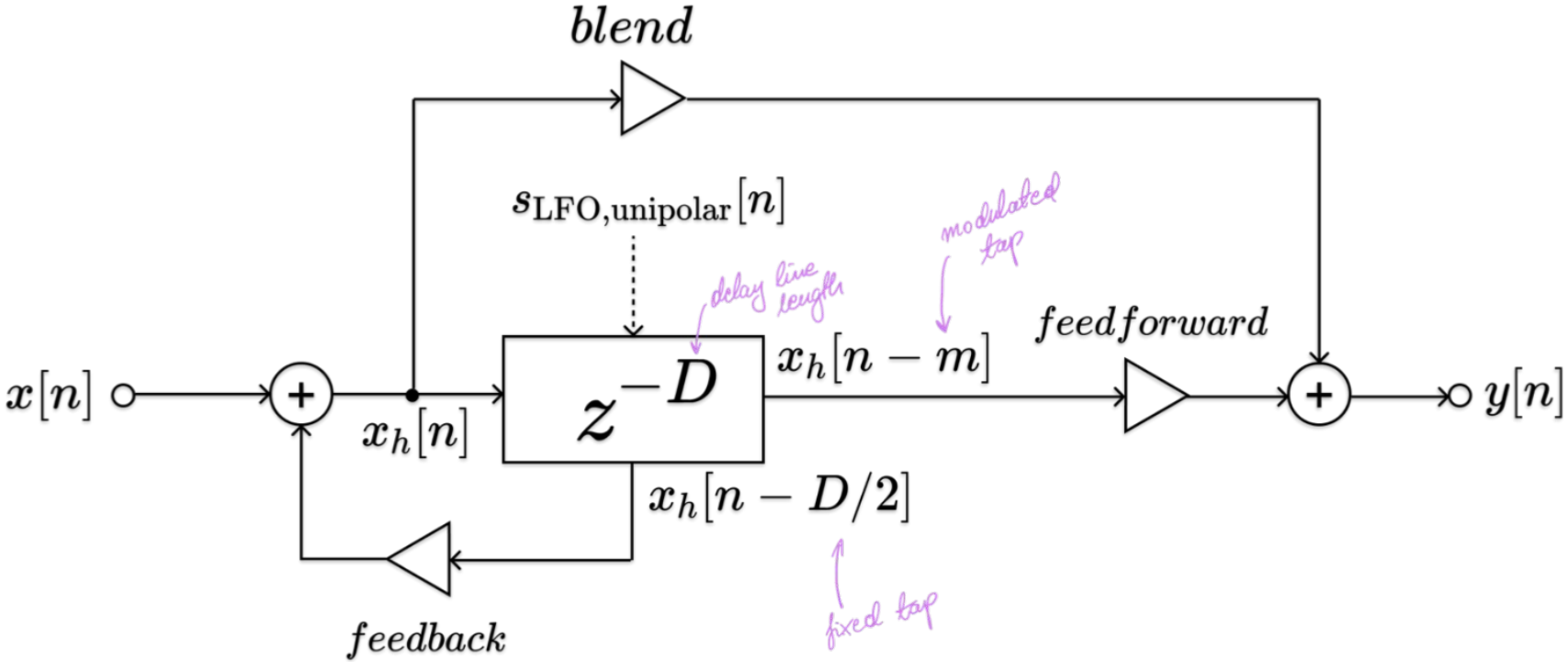


bipolar LFO



unipolar LFO

Task 1: Write down flanger update equations



Update equations

3. Output sample

$$y[n] = \text{blend } x_h[n] + \text{feedforward } x_h[n - m]$$

2. Helper sample

$$x_h[n] = x[n] + \text{feedback } x_h[n - D/2]$$

1. Modulated-delay value

$$m = s_{\text{LFO,unipolar}}[n]D$$

Prototype

- Goal: Quick creation of the desired effect for validation
- Technologies used:
  - Graphical
    - PureData
    - Max/MSP
  - Interpreted
    - Python
    - Matlab
  - Domain-specific languages (DSLs)
    - Faust
    - SuperCollider
  - ?

Task 2: Prototype

1. `git checkout task2a`
2. Run `python py/main.py data/saw200.0Hz5.0s.wav` and check that audibly the output signal is the same as the input signal
3. Inspect the spectrograms generated in the *output* folder. Are they identical visually?
4. Properly initialize the `Flanger` class instance
5. Implement flanger update equations without the LFO in `process_sample()` in *flanger.py*
6. Add an LFO

Surprise 1

Break

Until 3:15 PM

Implementation (the DSP part)

- Typically done in C++ with JUCE
- Other options:
  - C?
  - JavaScript?
  - Rust?
  - ?

Implementation (the DSP part)

- Start with mono first
- Disregard parameter smoothing
- Focus on correctness, ignore optimization
- Generic GUI
  - e.g., `juce::GenericAudioProcessorEditor`
- End-2-end verification tests

C++ implementation tips

- Use github.com/JanWilczek/audio-plugin-template
  - Treat warnings as errors
- **Keep the design flexible**
- Commit early, commit often (but double-check what you commit)
- Use end-2-end unit tests (e.g., `wolfsound::ProcessorFileIoTest`)
- **Unit-test what you can**
- Don’t be afraid of writing pseudocode
- Use `juce::juce_dsp`-like interface for easy composability
  - `process(const ProcessContext&)`
  - `prepare()`
  - `reset()`

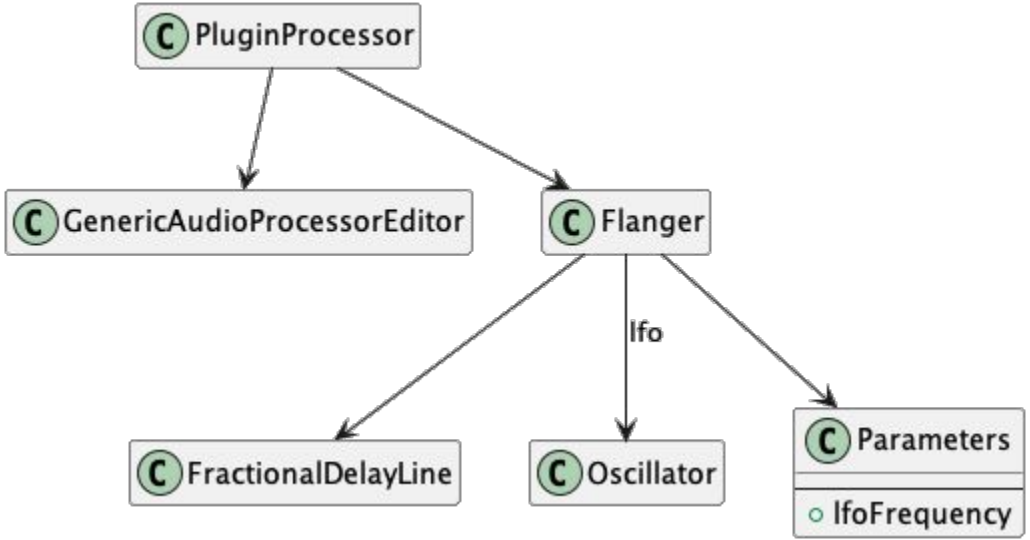
BUILDING A PLUGIN ASSEMBLY LINE

THE ROAD TO RAPID PLUGIN DEVELOPMENT

ADC24 Bristol

BENCE KOVÁCS &
LINUS CORNELIUSSON

C++ implementation: class diagram



Task 3: Implementation part 1

1. `git checkout task3a`
2. Build the C++ project
3. Run the end-2-end test
4. Test in a DAW
5. Implement `Flanger` member functions
    a. constructor
    b. `reset()`
    c. `prepare()`
    d. `processSample()`
6. Make the end-2-end test produce the same result as fixed-delay Python code

Tweaking parameters/experimentation

- "dev" parameters and "user" parameters
  - dev parameters will be fixed before releasing the plugin and inaccessible to the user
- Explore!
  - Expand the "allowed" range of parameters
  - Make some parameters unrealistic
  - "Model bending"

Task 4: Implementation part 2

1. `git checkout task4a`
2. Add an LFO (e.g., `juce::dsp::Oscillator`) and check again the end-2-end test against Python output
3. Make parameters (LFO frequency) adjustable
   a. Create a `Flanger::Parameters` struct
   b. Create a setter for the parameters
   c. Set the parameters in `processBlock()`
   d. Create an `AudioProcessorValueTreeState`
   e. Add the LFO frequency parameter to the plugin
4. Return a `juce::GenericAudioProcessorEditor` in `PluginProcessor::createEditor()`
5. Test in a DAW
6. Make stereo using JUCE's `ProcessorDuplicator`

Field testing (beta testing, validation)

- Give the plugin to a representation of your user base
- In the worst case: test it just yourself

Iteration on the design/implementation

- Should we improve the design or the implementation?
  - Design 👉 back to the research phase
  - Implementation 👉 code tuning (optimization) according to well-defined metrics
- Establish approval metrics/criteria:
  - What is the desired behavior for a particular signal?
  - Is there a baseline that a particular use case should match?
  - Are any signal types problematic?
  - Are certain parameter configurations problematic?
  - Are certain signal elements problematic (e.g., transients, decays)?
  - How fast a piece of code should be?


MLC S_Zero 93
S_ZERO-9.5°
A B
Default Preset
BOGREN DIGITAL
IN
CAB
OUT
POWER
6L6
KT88
PRESENCE DEPTH
GATE
OFF ON
BASS MIDDLE TREBLE
GAIN
VOLUME MASTER
MODE
INPUT
S_ZERO-9.5°
SILENOZ SIGNATURE
REC
ULTRA-QUALITY

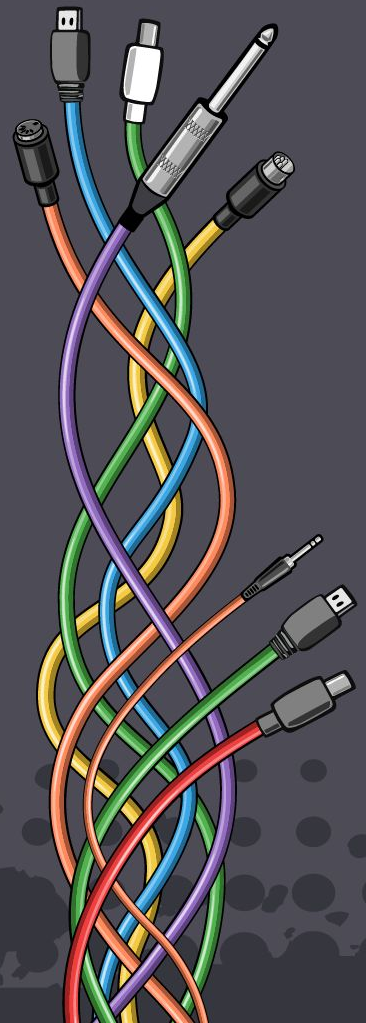

MLC S_Zero 93
S_ZERO-9.5°
A B
Default Preset
BOGREN DIGITAL
IN
CAB
OUT
OCTAVE 1
OCTAVE 2
DIRECT
FASCINATION STREET
OCTAVE
LEVEL
TONE
DISTORTION
MODE
MLC
BROWN SUGAR
LEVEL
TONE
DRIVE
FASCINATION STREET
OVD-2
REC
ULTRA-QUALITY

What we didn't cover

- Business side of plugin creation
  - Will my plugin sell?
  - Packaging for distribution
  - Selling
  - Distribution
- Code optimization (tuning)
- Presets
- Undo & redo
- …

Where to get more info on DSP & plugin dev

- WolfSound blog: TheWolfSound.com
- WolfSound YouTube: www.youtube.com/@WolfSoundAudio
- DSP Pro online course on learning digital signal processing for audio programming: wolfsoundacademy.com/dsp-pro
  - 20% special offer discount with promo code: ADC25_WORKSHOP (valid until December 10)
- Workshop feedback: jan.wilczek@thewolfsound.com



THE OFFICIAL JUCE AUDIO PLUGIN DEVELOPMENT COURSE IS HERE

AND IT'S FREE...

JAN WILCZEK & TOM POOLE

ADC25 Bristol

Where to get more info on DSP & plugin dev

- WolfSound blog: TheWolfSound.com
- WolfSound YouTube: www.youtube.com/@WolfSoundAudio
- DSP Pro online course on learning digital signal processing for audio programming: wolfsoundacademy.com/dsp-pro
  - 20% special offer discount with promo code: ADC25_WORKSHOP (valid until December 10)
- Workshop feedback: jan.wilczek@thewolfsound.com

"The venue team will now be transforming the conference space ready for the evening activities including the ADC Quiz, and therefore the main Bristol Suite will be unavailable.

Please use the next 1.5 hours for networking.

The bar in the Empire Lobby - downstairs from this room - and another bar in the registration area are now open and serving alcoholic and soft drinks.

Alternatively, go through to the main hotel bar area via the shortcut at the back of this room (Conservatory) - please refer to the venue maps you were given at registration, or ask the volunteer team.

The JUCE team will be hanging out in the hotel bar talking all things JUCE and happy to take questions and discuss JUCE-related issues..

In general, please make use of the full space available.

An evening meal will be served in the Bristol Suite from 18.30 (6.30pm) and the ADC Quiz will follow around 19.00 (7pm)."