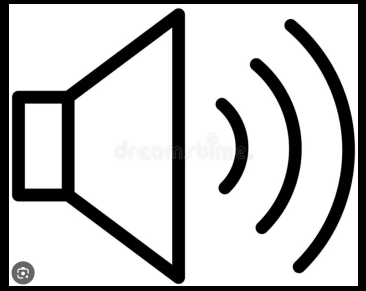
a stompbox in rust

Greg Travis
Enso Analytics

a stompbox in rust



why a stompbox?

?

why a stompbox?

Eventbrite

DIY Music Effects Workshop - Build a Stompbox with the Daisy Seed!

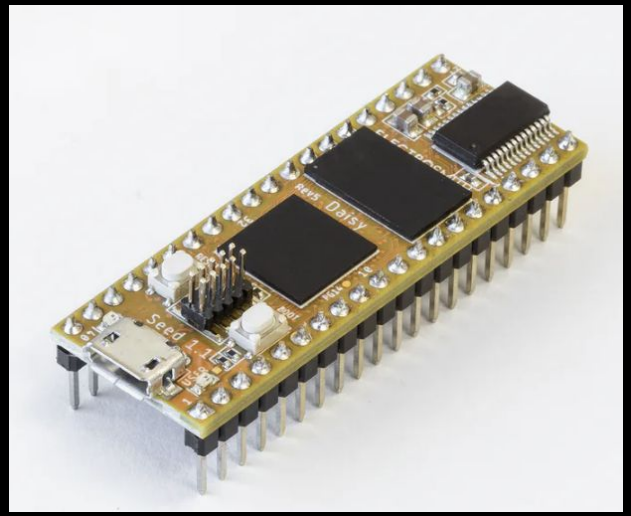
♬♫♪ Demystify music effects by building a custom, reprogrammable stomp box with the Daisy Seed! ♪♫♬

i took a class

why a stompbox?

i took a class

why a stompbox?



daisy seed

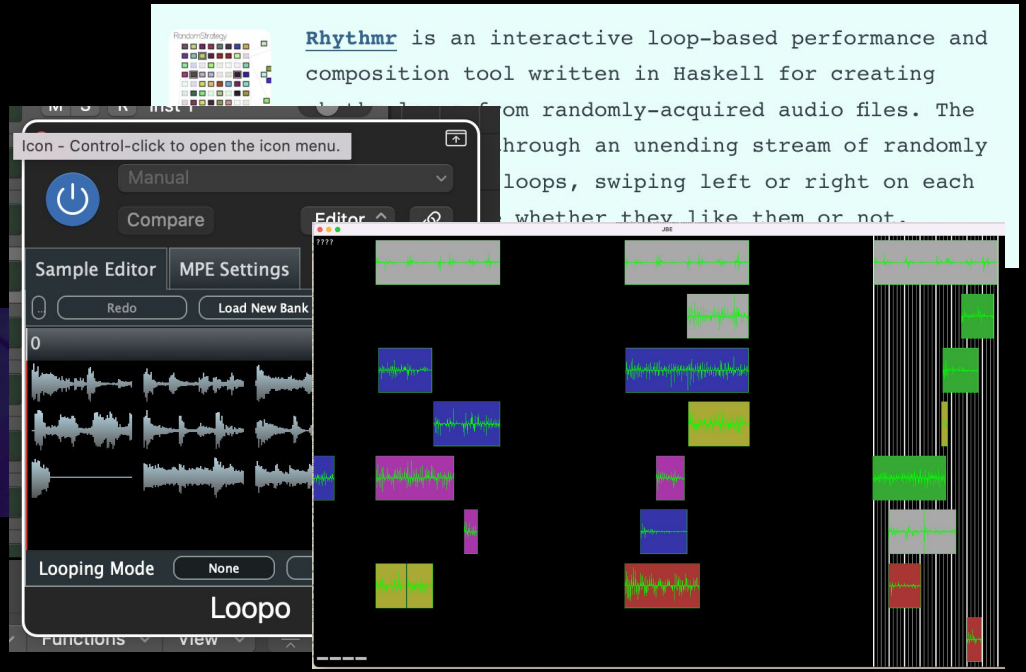
why a stompbox?

i like digital audio

why a stompbox?

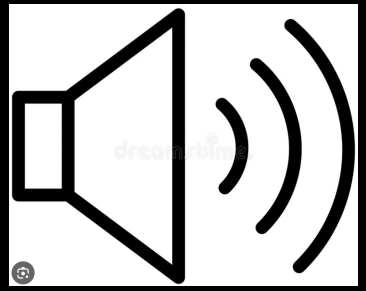
OBERLIN
COLLEGE & CONSERVATORY

TIMARA
TECHNOLOGY IN MUSIC AND RELATED ARTS



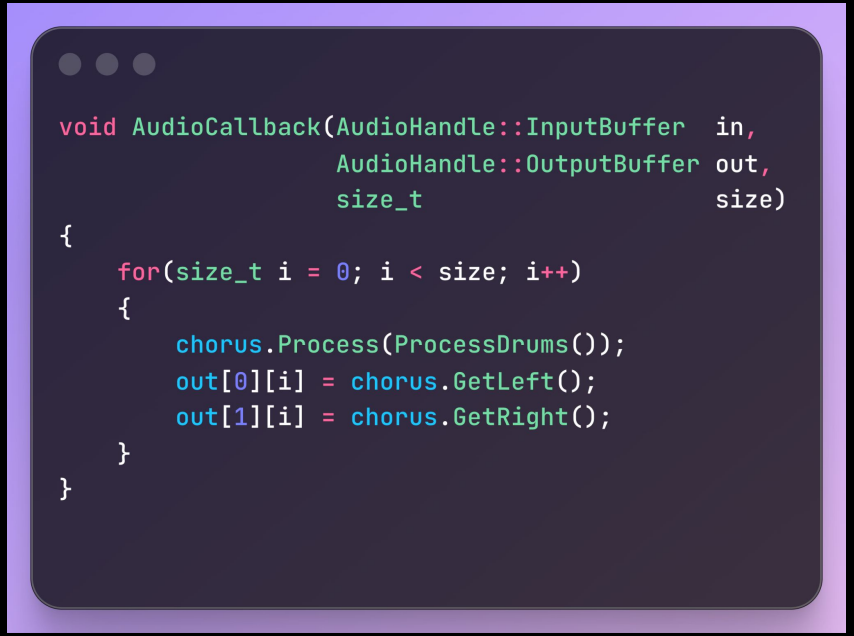
i like digital audio

why a stompbox?



simple

why a stompbox?



```
void AudioCallback(AudioHandle::InputBuffer  in,
                   AudioHandle::OutputBuffer out,
                   size_t                    size)
{
    for(size_t i = 0; i < size; i++)
    {
        chorus.Process(ProcessDrums());
        out[0][i] = chorus.GetLeft();
        out[1][i] = chorus.GetRight();
    }
}
```

simple

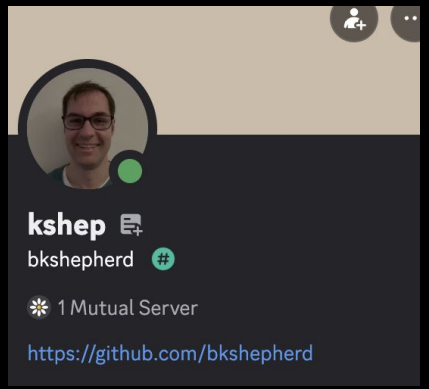
why a stompbox?

i like low-level code

why not a stompbox?

i ~~fear~~ don’t love soldering

why a stompbox?



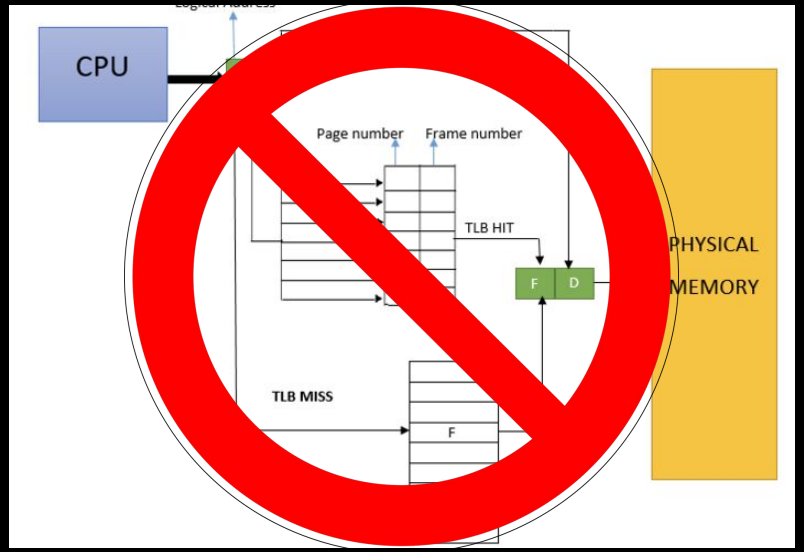
@kshep (daisy seed discord)

why rust?

why rust?

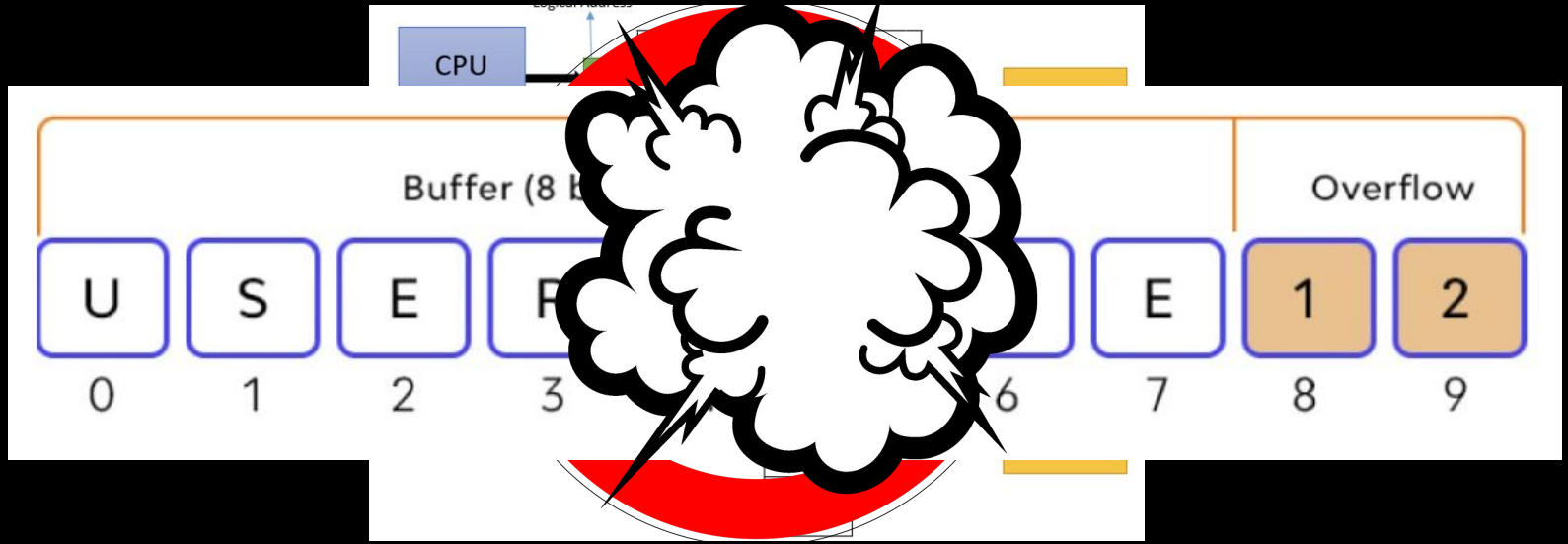
debugging is hard

why rust?



no mmu

why rust?



memory bugs

why rust?

let’s just not have bugs!

why rust?

spoiler: i had lots of bugs

Why Rust?

Use After Free

Stack Overflow

Interrupt Storm

```
0x2000000c
0x2000002c
0x2000004c
0x20000004
123456
4
0x2000002c
0x2000004c
0x2000006c
0x20000004
```

```
92ed031a  vldr   s2, [r2, #12]
d3ed031a  vldr   s3, [r3, #12]
92ed042a  vldr   s4, [r2, #16]
a6ee870a  vfma.f32       s0, s13
d3ed042a  vldr   s5, [r3, #16]
92ed053a  vldr   s6, [r2, #20]
d3ed053a  vldr   s7, [r3, #20]
92ed064a  vldr   s8, [r2, #24]
d3ed064a  vldr   s9, [r3, #24]
```

spoiler: i had lots of bugs

why rust?

Use A
Free

low

```
0x2000000c
0x2000002c
0x2000004c
0x20000004
123456
4
0x2000002c
0x2000004c
0x2000006c
0x20000004
```

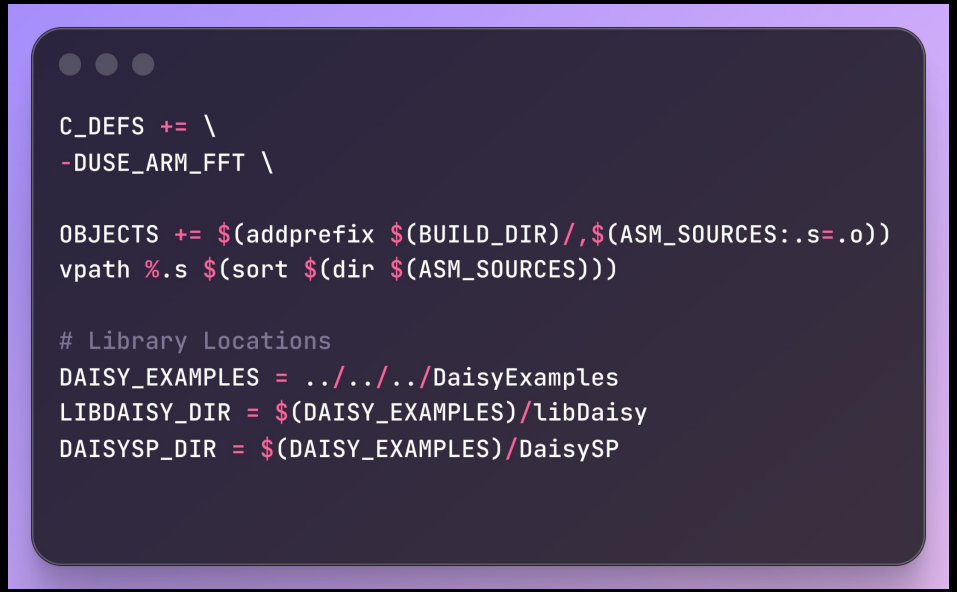
```
31a  vldr   s2, [r2, #12]
31a  vldr   s3, [r3, #12]
42a  vldr   s4, [r2, #16]
70a  vfma.f32       s0, s13
42a  vldr   s5, [r3, #16]
53a  vldr   s6, [r2, #20]
53a  vldr   s7, [r3, #20]
64a  vldr   s8, [r2, #24]
64a  vldr   s9, [r3, #24]
```

spoiler: i had lots of bugs

always take your headphones off
when flashing

why not rust?

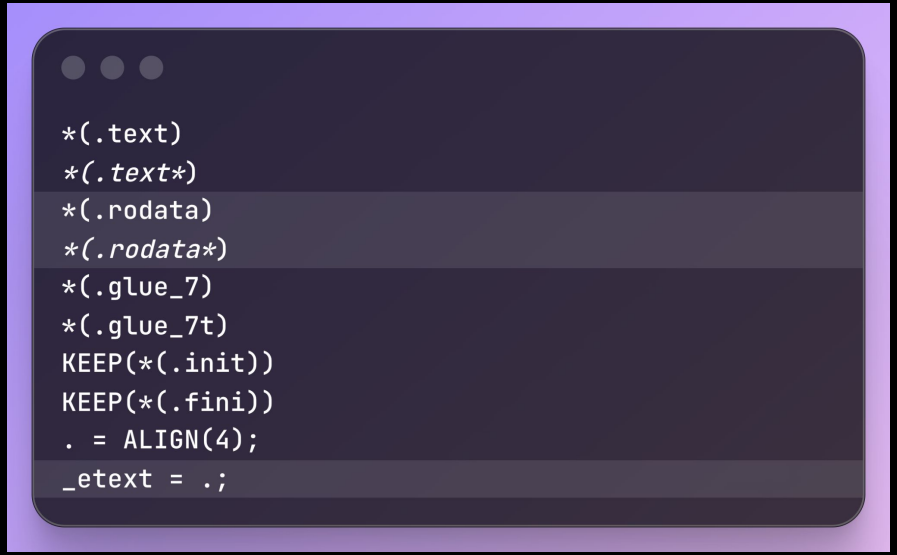
why not rust?



```
C_DEFS += \
-DUSE_ARM_FFT \

OBJECTS += $(addprefix $(BUILD_DIR)/,$(ASM_SOURCES:.s=.o))
vpath %.s $(sort $(dir $(ASM_SOURCES)))

# Library Locations
DAISY_EXAMPLES = ../../../DaisyExamples
LIBDAISY_DIR = $(DAISY_EXAMPLES)/libDaisy
DAISYSP_DIR = $(DAISY_EXAMPLES)/DaisySP
```

libDaisy makefile

why not rust?



```
*(.text)
*(.text*)
*(.rodata)
*(.rodata*)
*(.glue_7)
*(.glue_7t)
KEEP(*(.init))
KEEP(*(.fini))
. = ALIGN(4);
_etext = .;
```

linker scripts

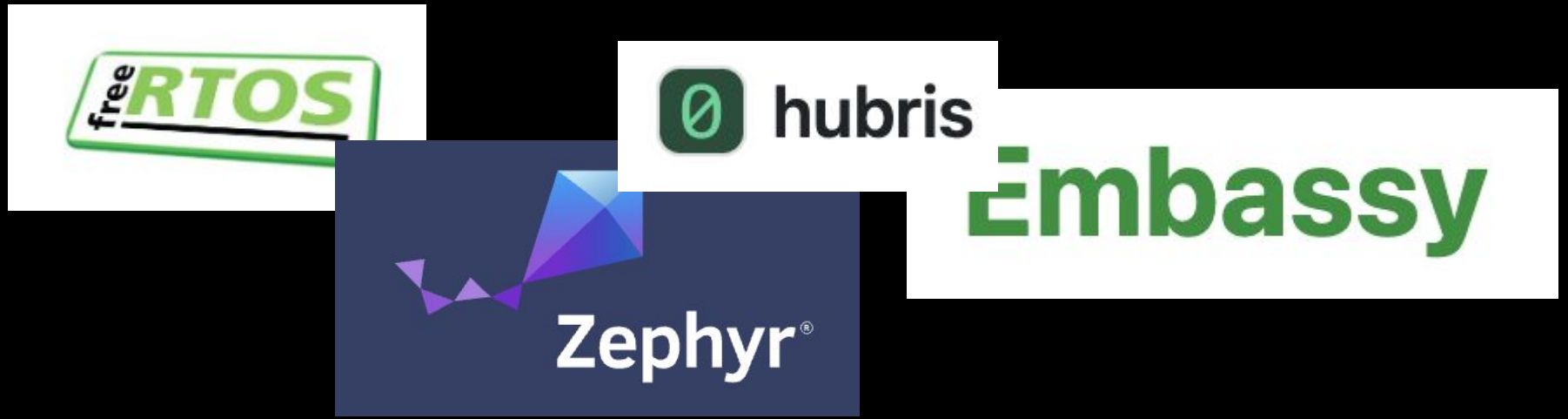
why not rust?

```
LIBDIR = -L ../../board/target/thumbv7em-none-eabihf/release
LIBS = -lpedalboard
```

rust library!

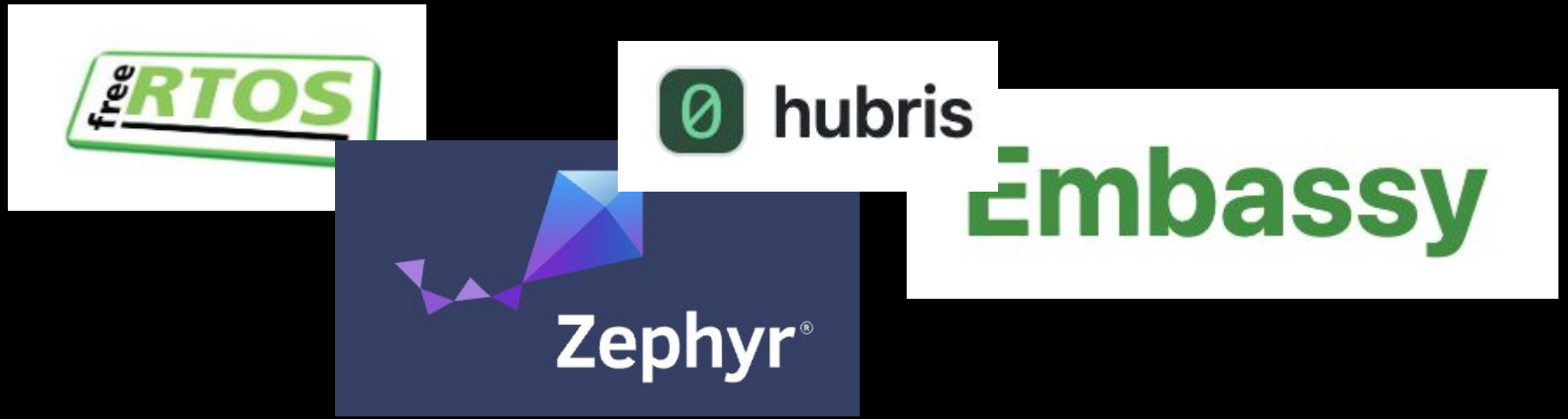
embedded rust

embedded rust



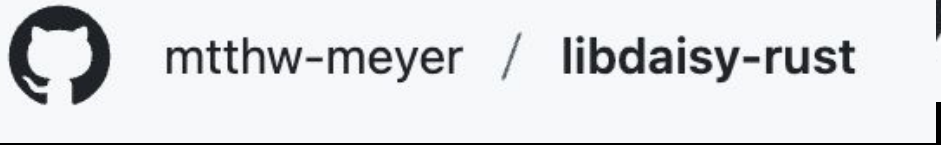
lots of frameworks

embedded rust



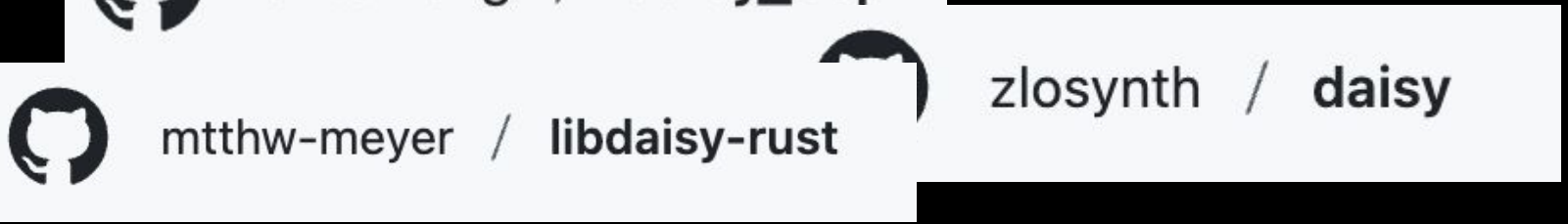
i didn’t use them

embedded rust



lots of open-source libraries

embedded rust

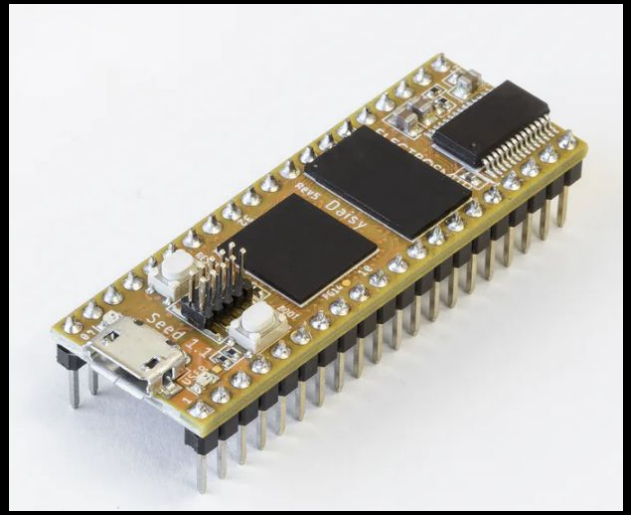


i didn’t use them

embedded rust

i didn’t use them

embedded rust

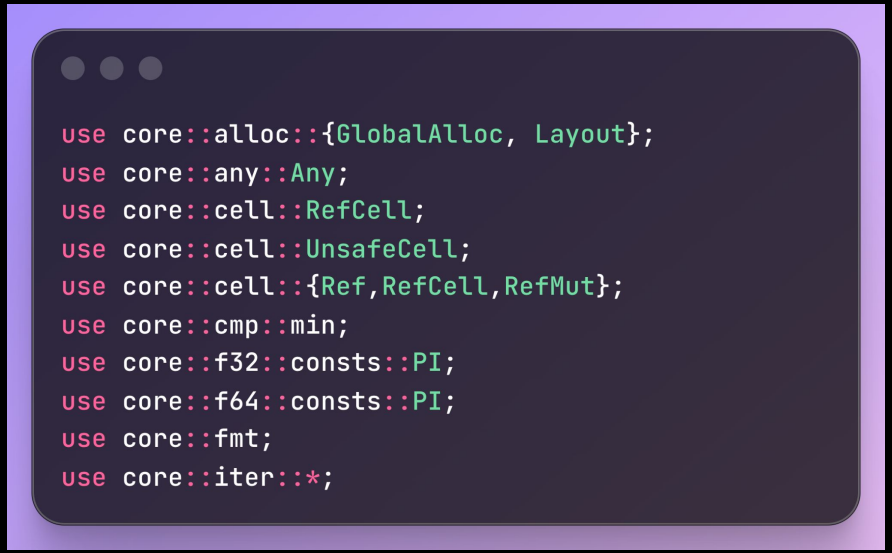


too new?

embedded rust

no_std

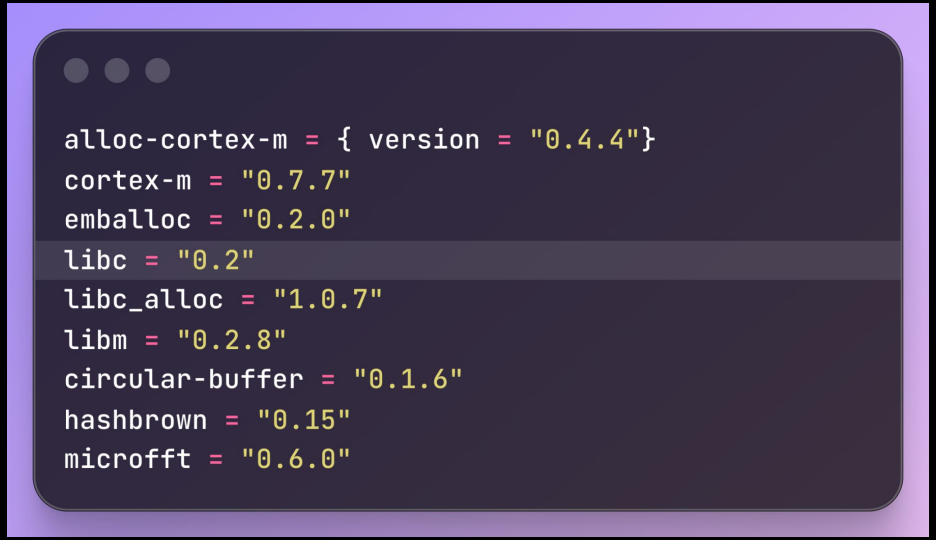
embedded rust



```rust
use core::alloc::{GlobalAlloc, Layout};
use core::any::Any;
use core::cell::RefCell;
use core::cell::UnsafeCell;
use core::cell::{Ref,RefCell,RefMut};
use core::cmp::min;
use core::f32::consts::PI;
use core::f64::consts::PI;
use core::fmt;
use core::iter::*;
```

whatever you need

embedded rust



```
alloc-cortex-m = { version = "0.4.4"}
cortex-m = "0.7.7"
emballoc = "0.2.0"
libc = "0.2"
libc_alloc = "1.0.7"
libm = "0.2.8"
circular-buffer = "0.1.6"
hashbrown = "0.15"
microfft = "0.6.0"
```

i did use them

the framework

the framework

```
void AudioCallback(AudioHandle::InputBuffer  in,
                   AudioHandle::OutputBuffer out,
                   size_t                    size)
{
    for(size_t i = 0; i < size; i++)
    {
        chorus.Process(ProcessDrums());
        out[0][i] = chorus.GetLeft();
        out[1][i] = chorus.GetRight();
    }
}
```

simple

the framework

```rust
impl Patch for LowPassFilter {
    fn rust_process_audio(
        &mut self,
        input_slice: &[f32],
        output_slice: &mut [f32],
        _knobs: &Box<dyn Knobs>,
        _playhead: Playhead,
    ) {
        for i in 0..input_slice.len() {
            output_slice[i] = 5.0 * ((input_slice[i] + self.state) / 2.0);
            self.state = input_slice[i];
        }
    }
}
```

Patch

the framework

```rust
pub struct Rig {
    pub patch: Box<dyn Patch>,
    pub knobs: Box<dyn Knobs>,
    pub toggle: Toggle,
    // ...
    pub framesize: usize,
    pub playhead: Playhead,
}
```

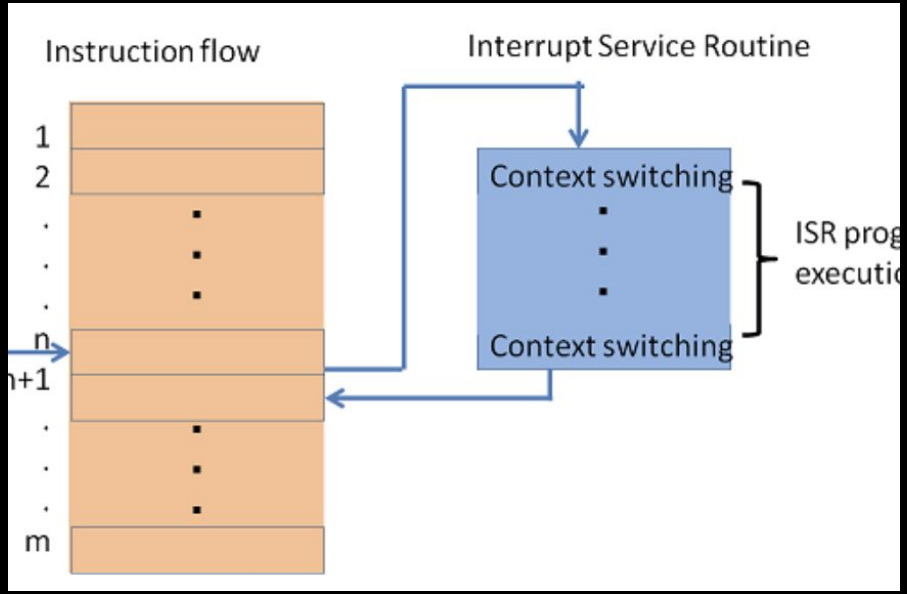
Patch

the framework

```rust
pub fn rig_install_callback();
pub fn rig_install_patch(box_patch: Box<dyn Patch>, knobs: Box<dyn Knobs>, toggle: Toggle);
pub fn rig_deinstall_patch() -> Option<Box<dyn Patch>>;
pub fn rig_run_patch_on_buffer(patch: Box<dyn Patch>, input: &[f32], output: &mut [f32]);
```

Rig

the framework



interrupts are scary

the framework

```
void AudioCallback(AudioHandle::InputBuffer in, AudioHandle::OutputBuffer out, size_t size)
{
  rig_process_audio_callback(in, out, size);
}

extern "C" void cpp_rig_install_callback()
{
  hw.StartAudio(AudioCallback);
}
```

interrupts are scary

the framework

is it a thread?

interrupts are scary

the framework

it’s kind of a thread

the framework

```rust
static THE_PATCH: Mutex<RefCell<Option<Rig>>> = Mutex::new(RefCell::new(None));

pub fn rig_set(rig: Rig) {
    interrupt::free(|cs| THE_PATCH.borrow(cs).replace(Some(rig)));
}

pub fn rig_clear() {
    interrupt::free(|cs| {
            THE_PATCH.borrow(cs).replace(None);
    });
}

pub fn rig_use<F>(f: F)
where
    F: FnOnce(&mut Rig) {
    interrupt::free(|cs| {
        if let Some(ref mut rig) = THE_PATCH.borrow(cs).borrow_mut().deref_mut().as_mut() {
            f(rig);
        }
    });
}

pub fn rig_install_callback() {
    unsafe {
        cpp_rig_install_callback();
    }
}
```

board

```rust
lazy_static! {
    static ref THE_PATCH: Mutex<Option<Rig>> = Mutex::new(None);
}

pub fn rig_set(rig: Rig) {
    *(THE_PATCH.lock().unwrap()) = Some(rig);
}

pub fn rig_clear() {
    *(THE_PATCH.lock().unwrap()) = None;
}

pub fn rig_use<F>(f: F)
where
    F: FnOnce(&mut Rig) {
    if let Some(ref mut rig) = *(THE_PATCH.lock().unwrap()) {
        f(rig);
    }
}

pub fn rig_install_callback() {
    let _handler = thread::spawn(|| {
        let input: [f32; BLOCK_SIZE] = [0.0; BLOCK_SIZE];
        let mut output: [f32; BLOCK_SIZE] = [0.0; BLOCK_SIZE];
        loop {
            rust_process_audio_soft(&input, &mut output, BLOCK_SIZE);
        }
    });
}
```

host

unportable

the framework

```rust
use core::cell::RefCell;
use core::ops::DerefMut;
use cortex_m::interrupt::{self, Mutex};

pub struct Globby<T> {
  thing: Mutex<RefCell<Option<T>>>,
}
```

board

```rust
extern crate std;

use std::sync::Mutex;

pub struct Globby<T> {
  thing: Mutex<Option<T>>,
}
```

host

less unportable

the framework

```rust
pub fn use_and_return<F, R>(&self, f: F) -> R
where
    F: FnOnce(&mut Option<T>) -> R {
    interrupt::free(|cs| {
        let mut binding = self.thing.borrow(cs).borrow_mut();
        let thing: &mut Option<T> = binding.deref_mut();
        f(thing)
    })
}
```

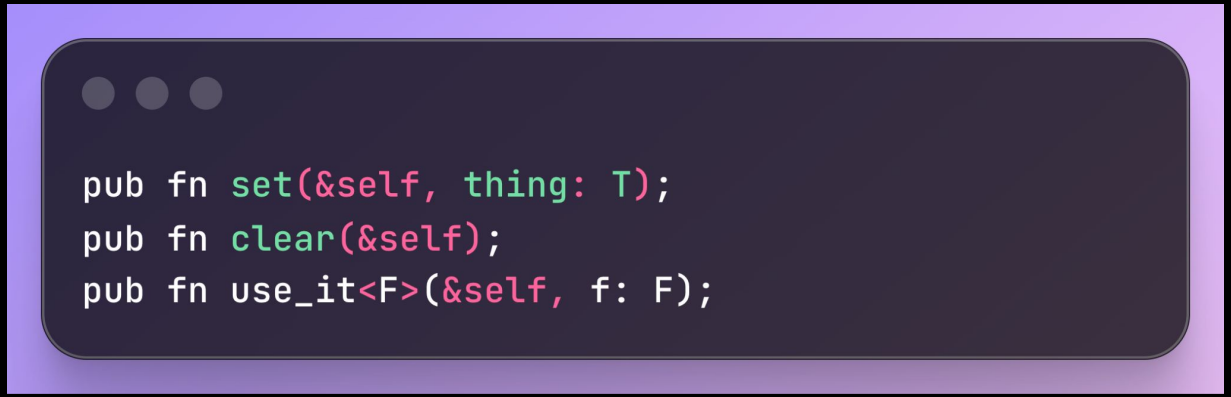
board

```rust
pub fn use_and_return<F, R>(&self, f: F) -> R
where
    F: FnOnce(&mut Option<T>) -> R {
    f(&mut self.thing.lock().unwrap())
}
```

host

even less unportable

the framework

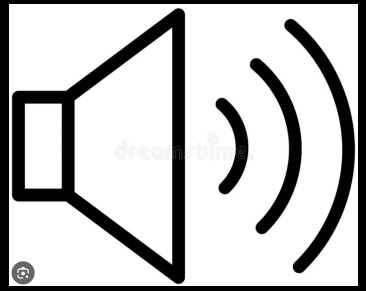


```
pub fn set(&self, thing: T);
pub fn clear(&self);
pub fn use_it<F>(&self, f: F);
```

both

kinda portable

framework features



more than audio flow

framework features

knobs

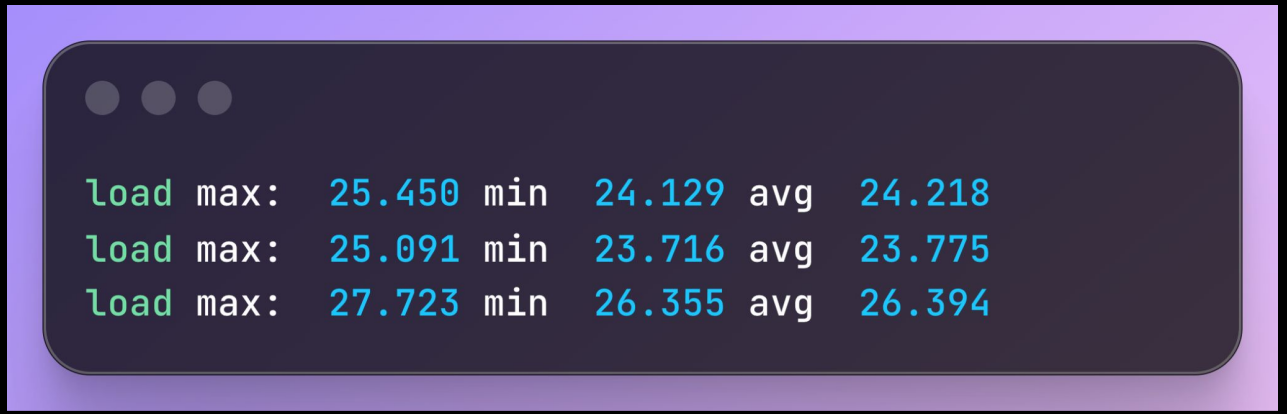
encoder

lights

switches

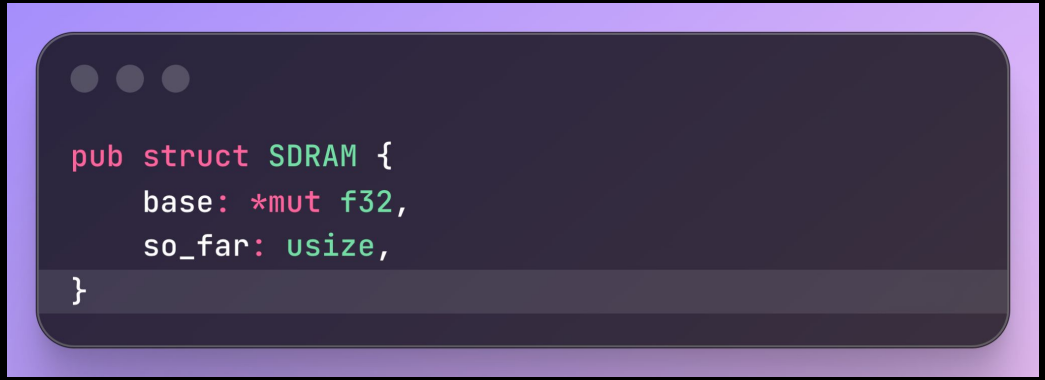
physical controls

framework features



load averages

framework features



```
pub struct SDRAM {
    base: *mut f32,
    so_far: usize,
}
```

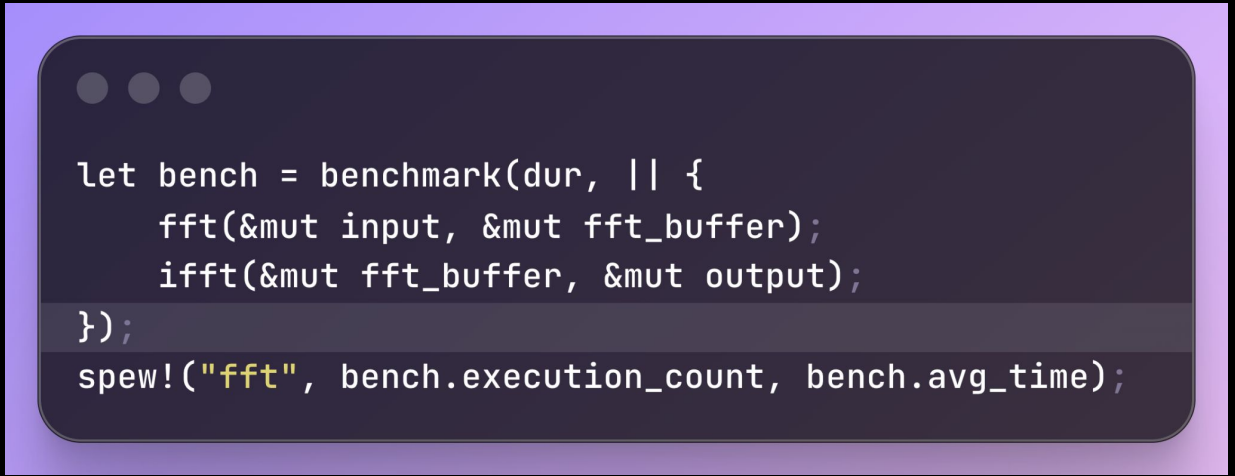
SDRAM allocator

framework features

```
spew!("buf_f", "r", r, "r_0", r_0, "r_1", r_1, "alpha", alpha, "beta", beta);
spew!("hmm", self.buf(12238), self.buf(12239), self.buf(12240));
spew!("ts", self.ts, "r", r, "w", w, "w_hat", w_hat, "t_r", t_r, "ramp",
    backward_ramp_start, backward_ramp_end, "alpha", a0, a1, alpha,
    "main out", self.buf_f(r), "alt out", self.buf_f(alt_r), "out", out);
```

logging

framework features



```
let bench = benchmark(dur, || {
    fft(&mut input, &mut fft_buffer);
    ifft(&mut fft_buffer, &mut output);
});
spew!("fft", bench.execution_count, bench.avg_time);
```

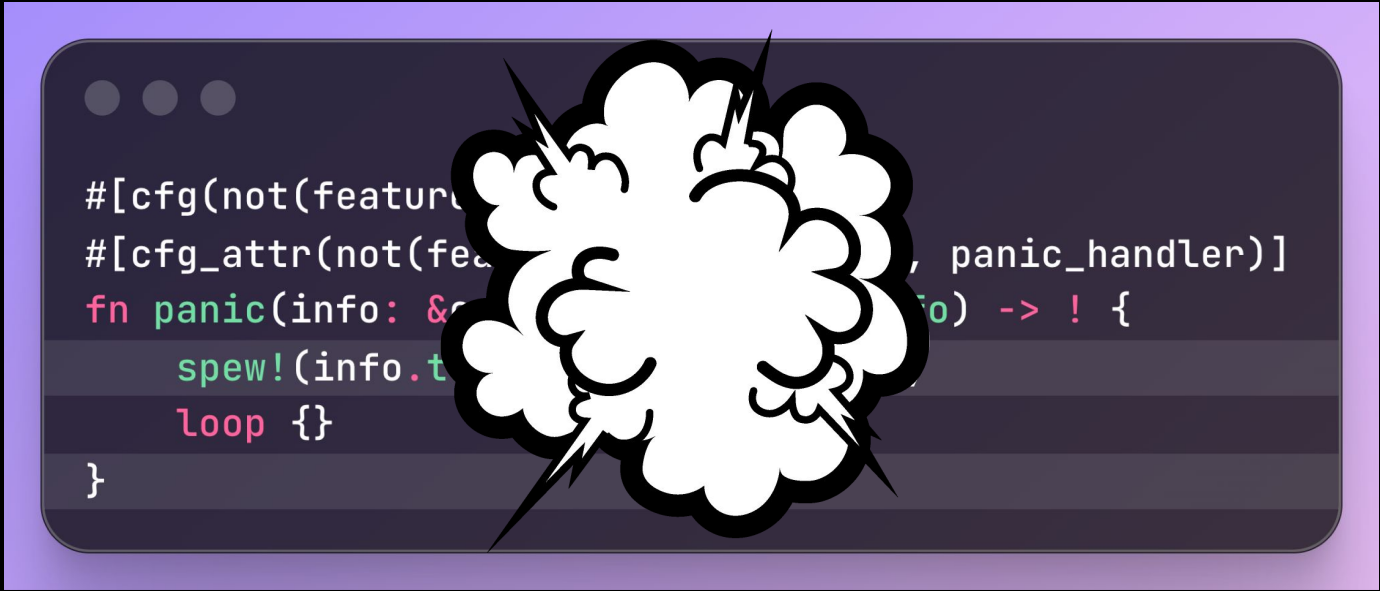
benchmarking

framework features

```
#[cfg(not(feature = "for_host"))]
#[cfg_attr(not(feature = "for_host"), panic_handler)]
fn panic(info: &core::panic::PanicInfo) -> ! {
    spew!(info.to_string().as_str());
    loop {}
}
```

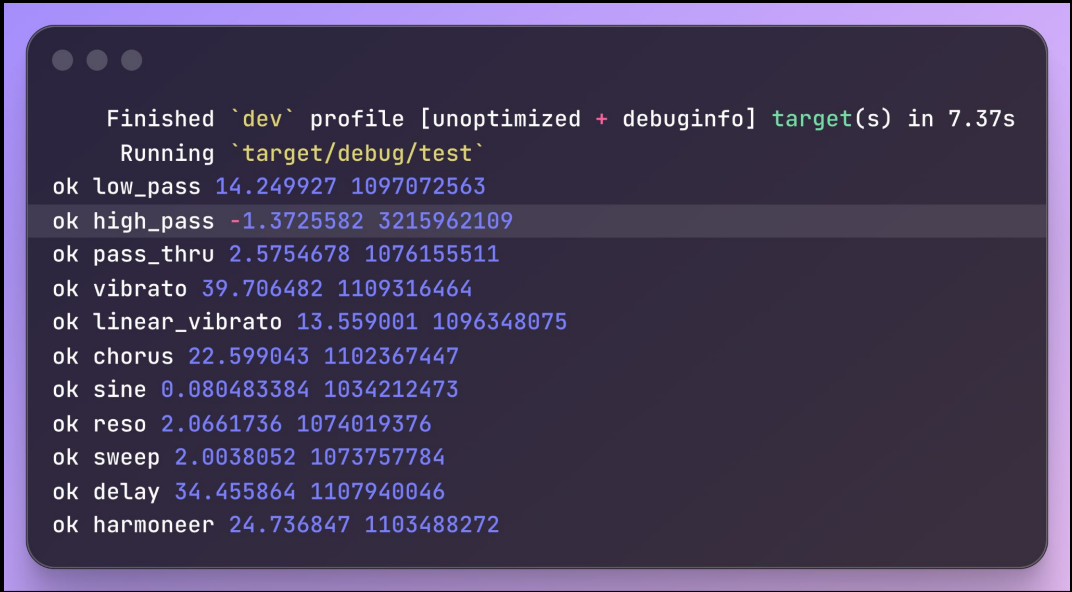
panic handler

framework features



panic handler

framework features

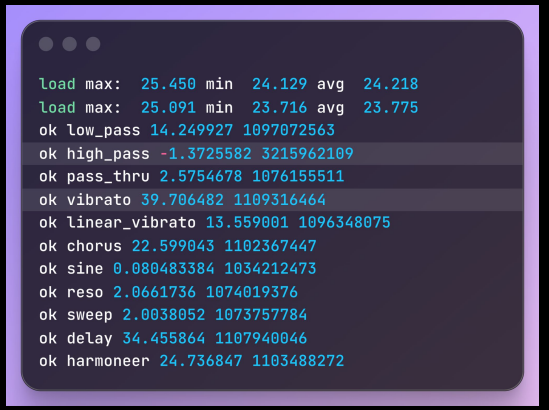


unit tests

framework features

```
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 7.37s
     Running `target/debug/test`
ok low_pass 14.249927 1097072563
ok high_pass -1.3725582 3215962109
ok pass_thru 2.5754678 1076155511
ok vibrato 39.706482 1109316464
ok linear_vibrato 13.559001 1096348075
ok chorus 22.599043 1102367447
ok sine 0.080483384 1034212473
ok reso 2.0661736 1074019376
ok sweep 2.0038052 1073757784
ok delay 34.455864 1107940046
ok harmoneer 24.736847 1103488272
```

board

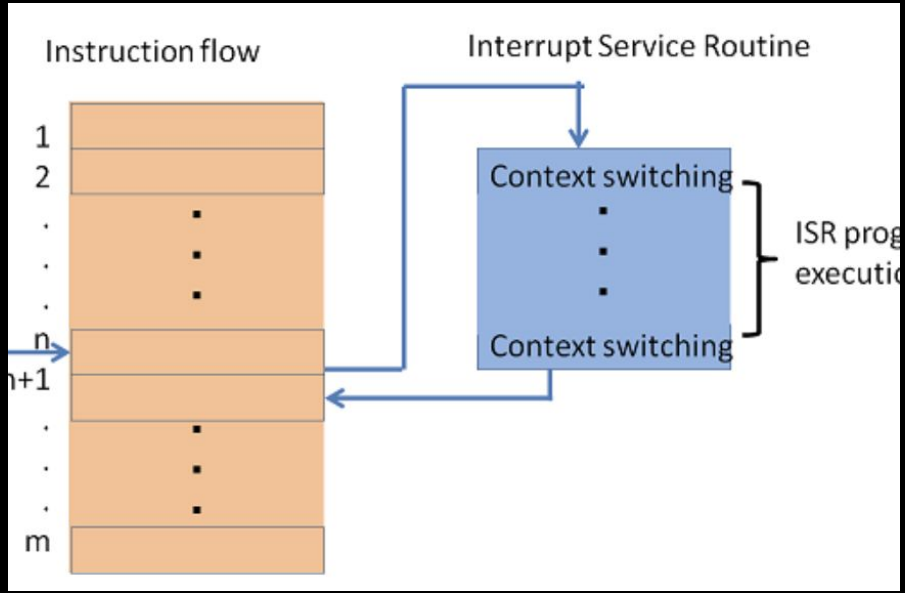


```
load max:  25.450 min  24.129 avg  24.218
load max:  25.091 min  23.716 avg  23.775
ok low_pass 14.249927 1097072563
ok high_pass -1.3725582 3215962109
ok pass_thru 2.5754678 1076155511
ok vibrato 39.706482 1109316464
ok linear_vibrato 13.559001 1096348075
ok chorus 22.599043 1102367447
ok sine 0.080483384 1034212473
ok reso 2.0661736 1074019376
ok sweep 2.0038052 1073757784
ok delay 34.455864 1107940046
ok harmoneer 24.736847 1103488272
```

host

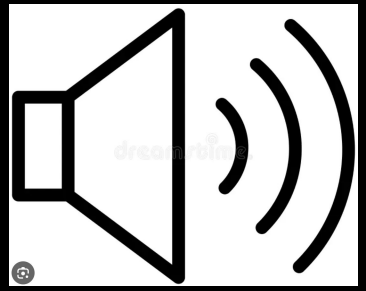
unit tests

framework features



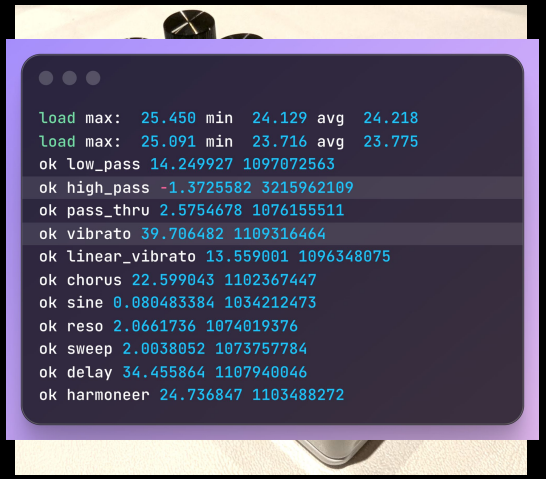
testing inside the interrupt

framework features

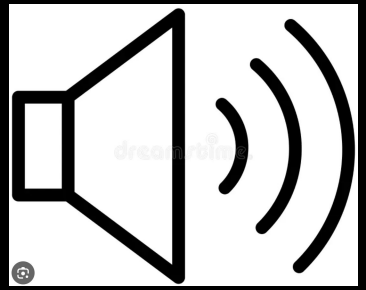


live unit tests

framework features



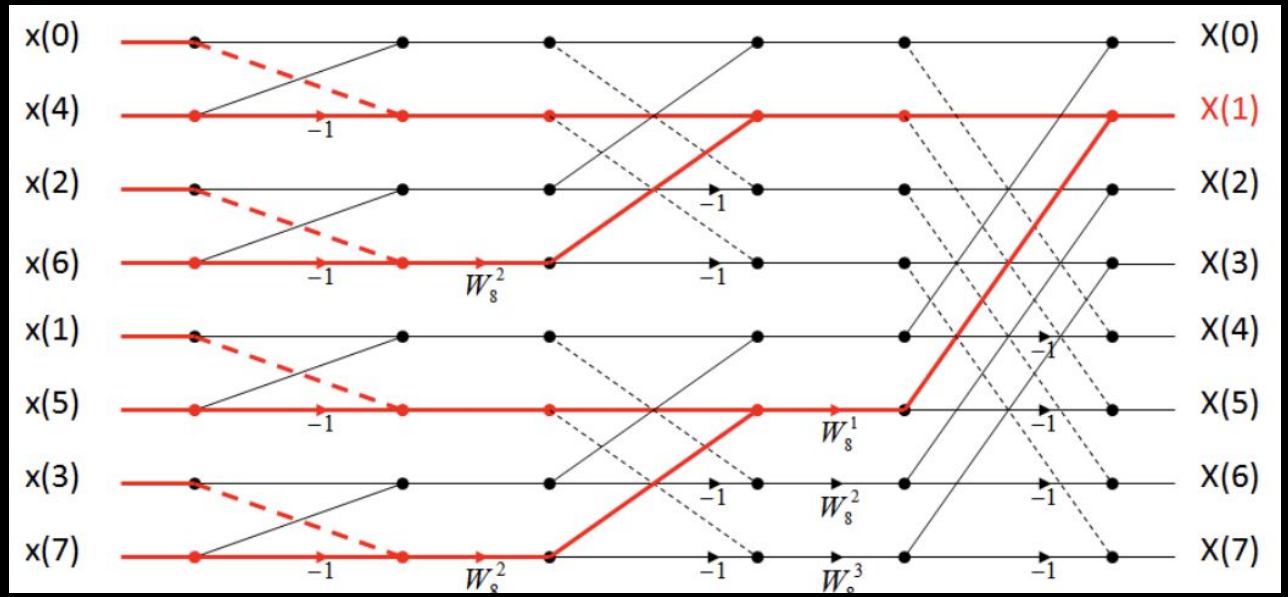
```
load max:  25.450 min  24.129 avg  24.218
load max:  25.091 min  23.716 avg  23.775
ok low_pass 14.249927 1097072563
ok high_pass -1.3725582 3215962109
ok pass_thru 2.5754678 1076155511
ok vibrato 39.706482 1109316464
ok linear_vibrato 13.559001 1096348075
ok chorus 22.599043 1102367447
ok sine 0.080483384 1034212473
ok reso 2.0661736 1074019376
ok sweep 2.0038052 1073757784
ok delay 34.455864 1107940046
ok harmoneer 24.736847 1103488272
```



live unit tests

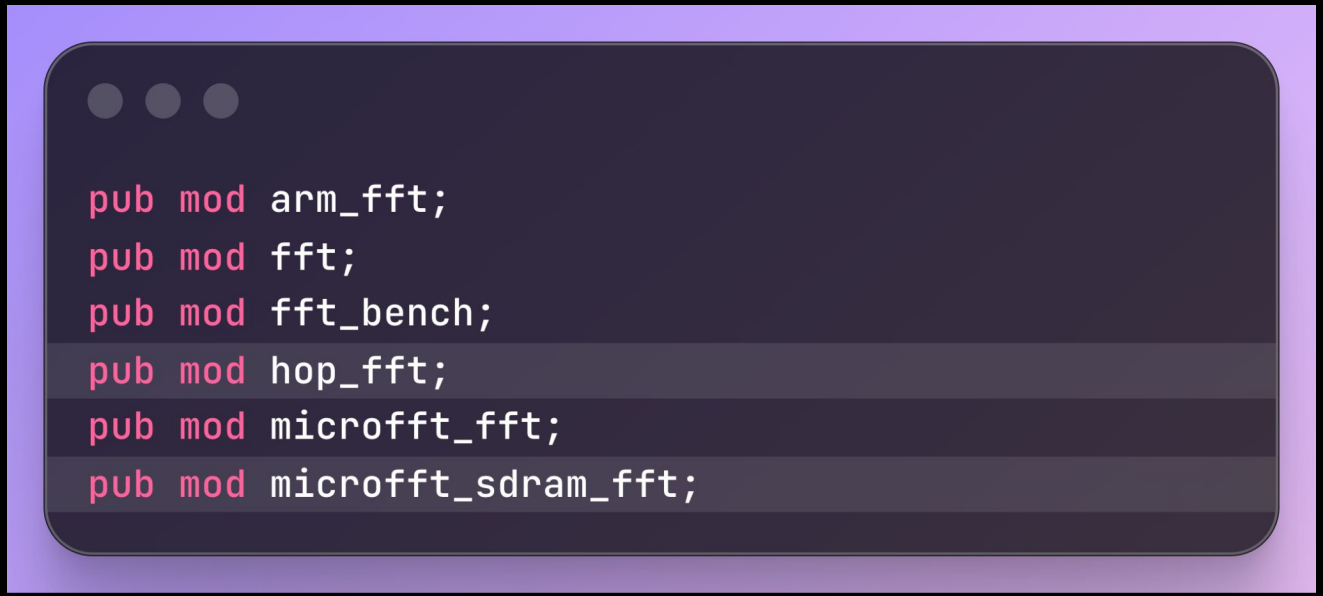
audio features

audio features



FFTs

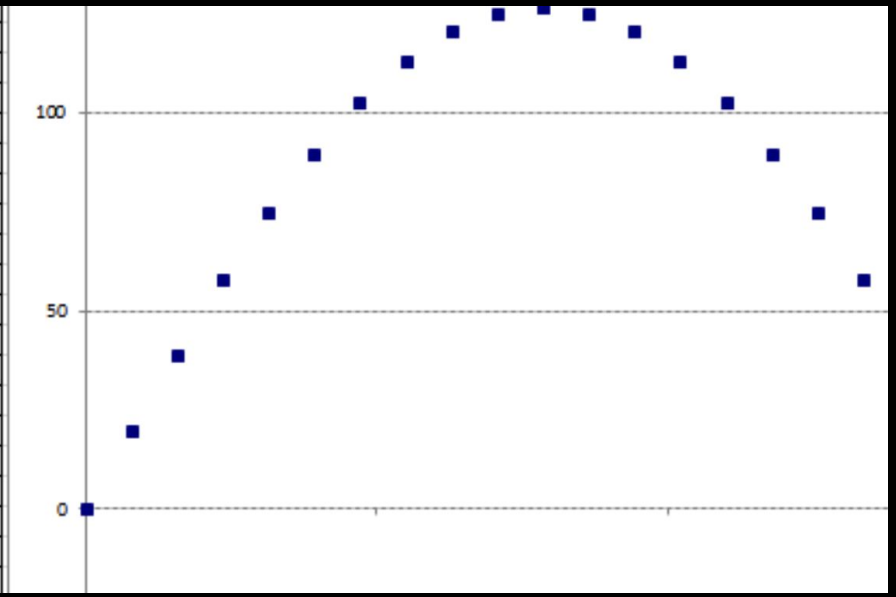
audio features



FFTs

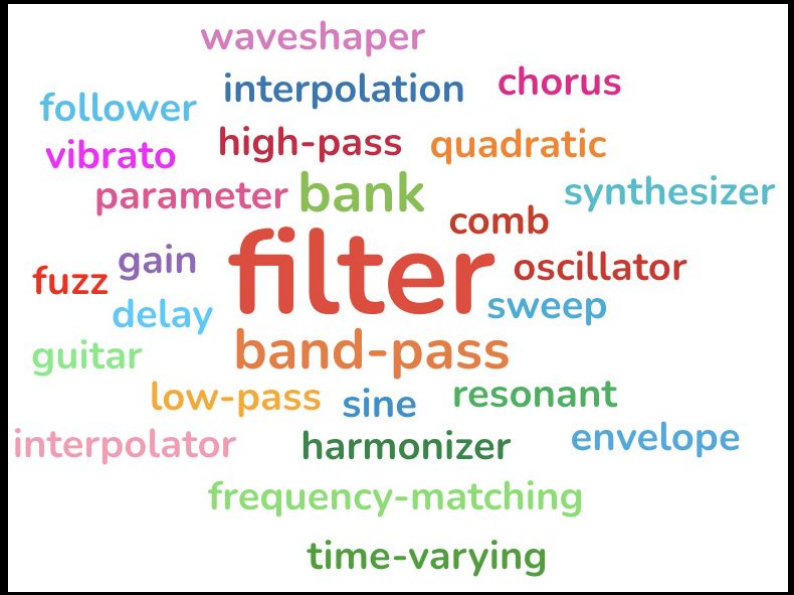
audio features

1010	3A
1011	4B
1010	5A
0111	67
0001	71
0001	79
1101	7D
1111	7F
1101	7D
0001	79
0001	71
0111	67
1010	5A
1011	4B
1010	3A
0111	27
0100	14
0000	0
0100	94



fast lookup table sin

audio features



audio effects

audio features

audio effects

audio features

the eventide h910 harmonizer (1975)

audio features

the eventide h910 harmonizer (1975)

audio features

"arguably the first pro audio digital effects product"

the eventide h910 harmonizer (1975)

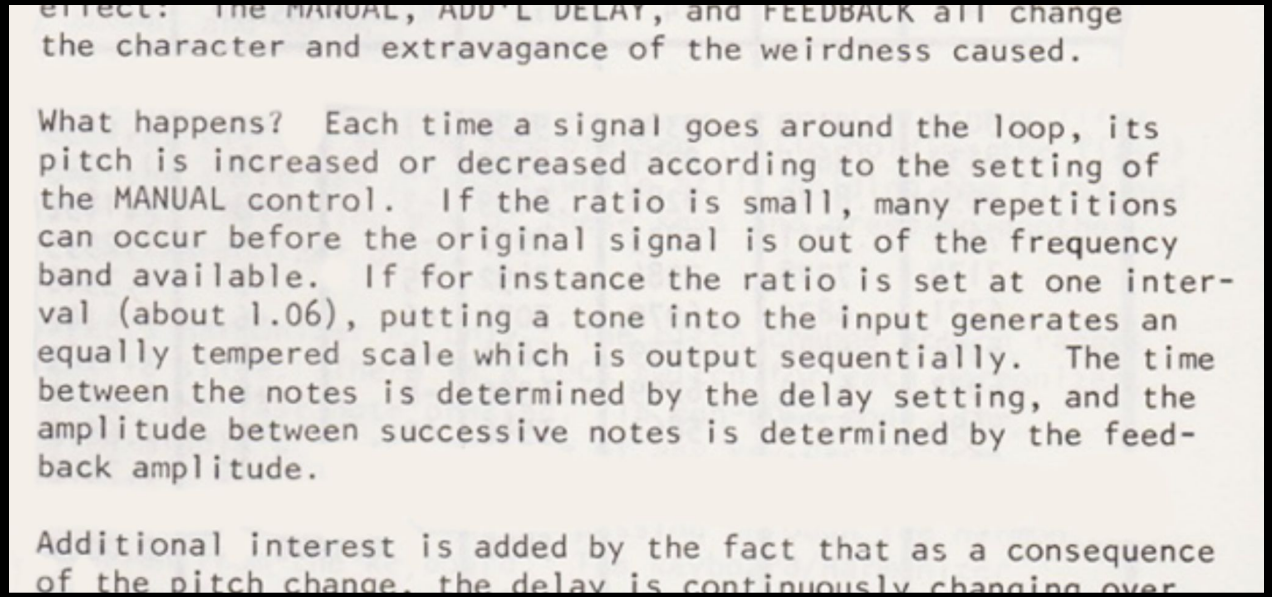
audio features

NAMM hall of fame (2007)

audio features

the eventide h910 harmonizer (1975)

audio features



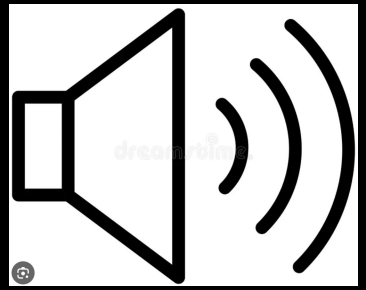
effect! The MANUAL, ADD'L DELAY, and FEEDBACK all change the character and extravagance of the weirdness caused.

What happens? Each time a signal goes around the loop, its pitch is increased or decreased according to the setting of the MANUAL control. If the ratio is small, many repetitions can occur before the original signal is out of the frequency band available. If for instance the ratio is set at one interval (about 1.06), putting a tone into the input generates an equally tempered scale which is output sequentially. The time between the notes is determined by the delay setting, and the amplitude between successive notes is determined by the feedback amplitude.

Additional interest is added by the fact that as a consequence of the pitch change, the delay is continuously changing over

the eventide h910 harmonizer (1975)

audio features



the harmoneer (2025)

the edsl

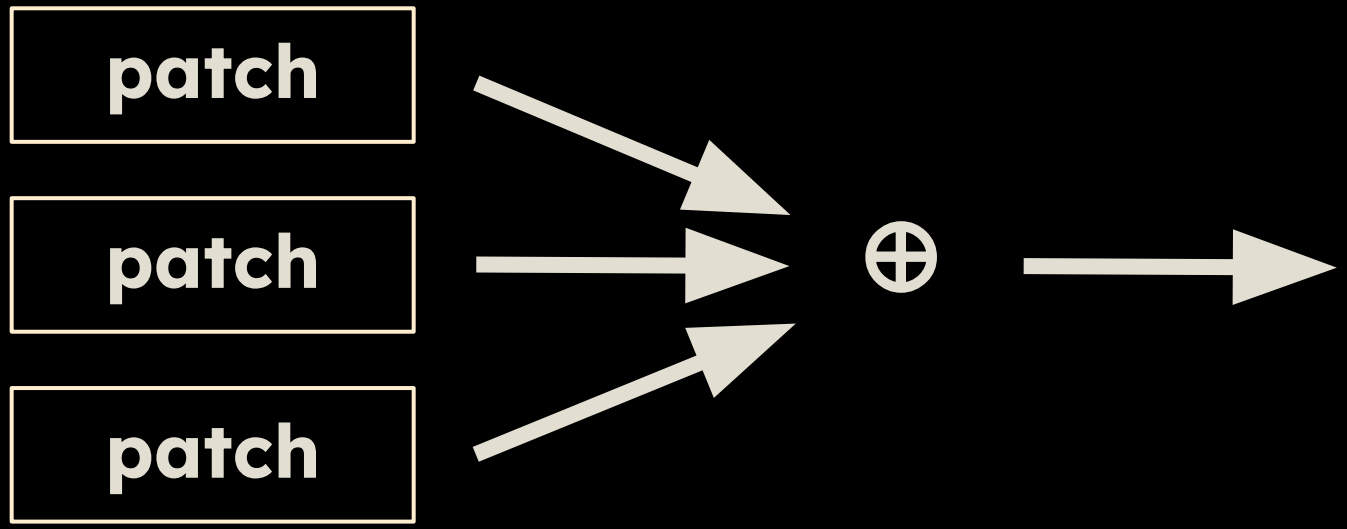
the edsl

simple topology

the edsl

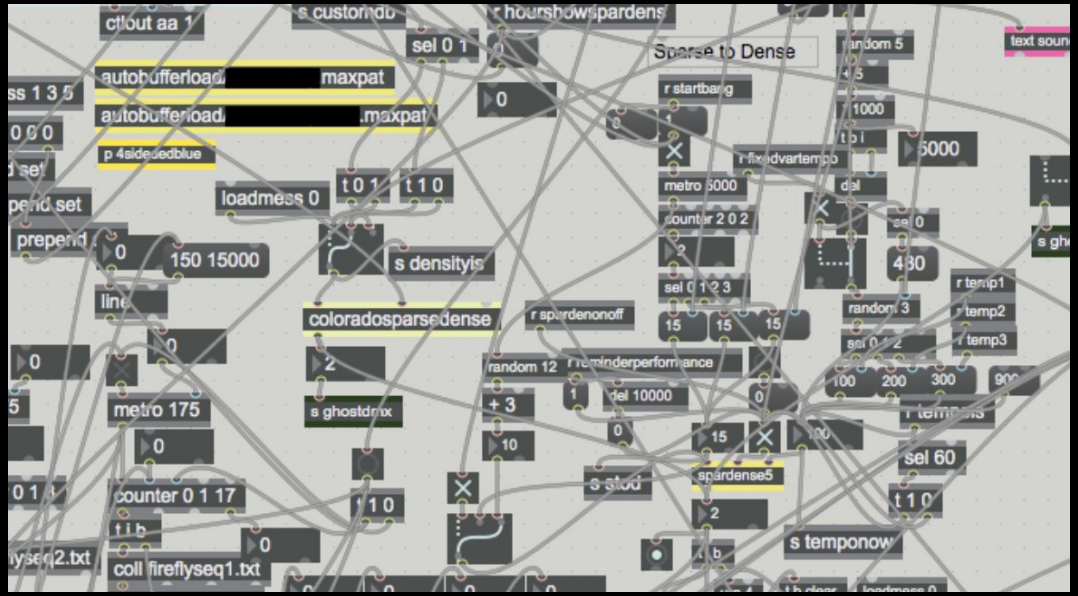
simple topology: Seq

the edsl



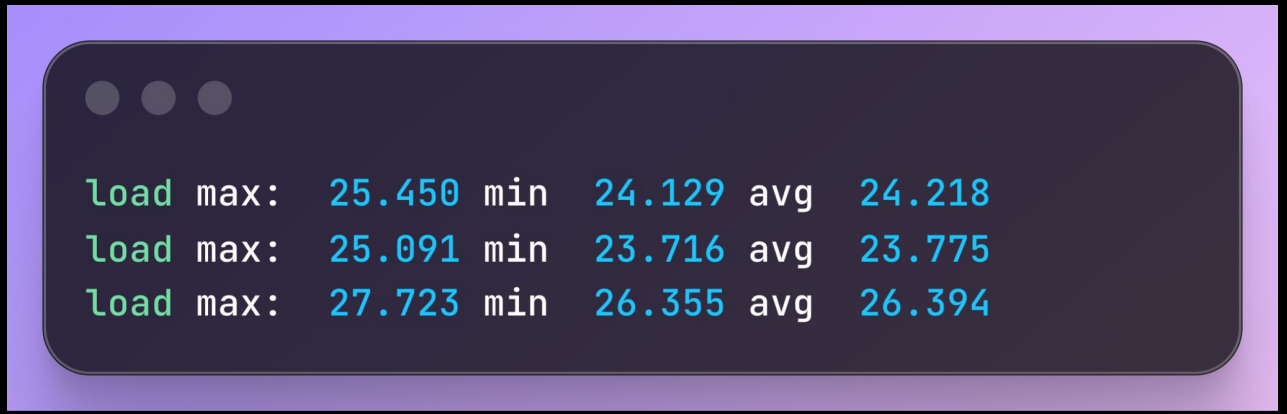
simple topology: Mixer

the edsl



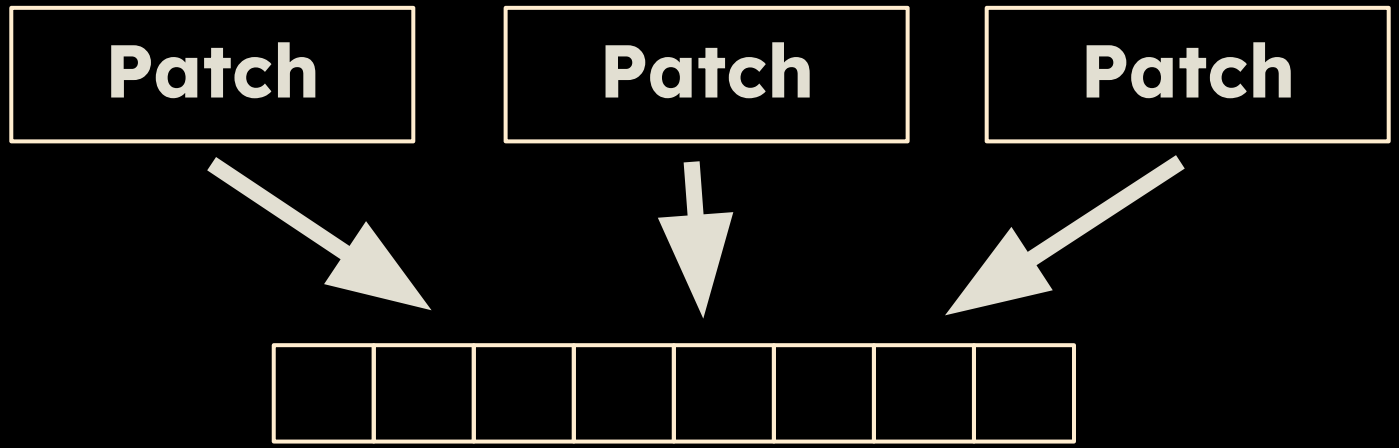
max (cycling ‘74)

the edsl



faster?

the edsl



share memory?

the edsl

```
fn build_edsl_nodey() {
    let input = input();
    let pt = pass_thru(&input);
    let added = add(&input, &pt);
    let sf = sum_filter(&added, -1, 1);
    let sf2 = sum_filter(&added, -3, 3);
    let sfadd = add(&sf, &sf2);
    let out = sfadd;
    compile(&out, "src/filter/edsl_nodey.rs", "EdslNodey");
}
```

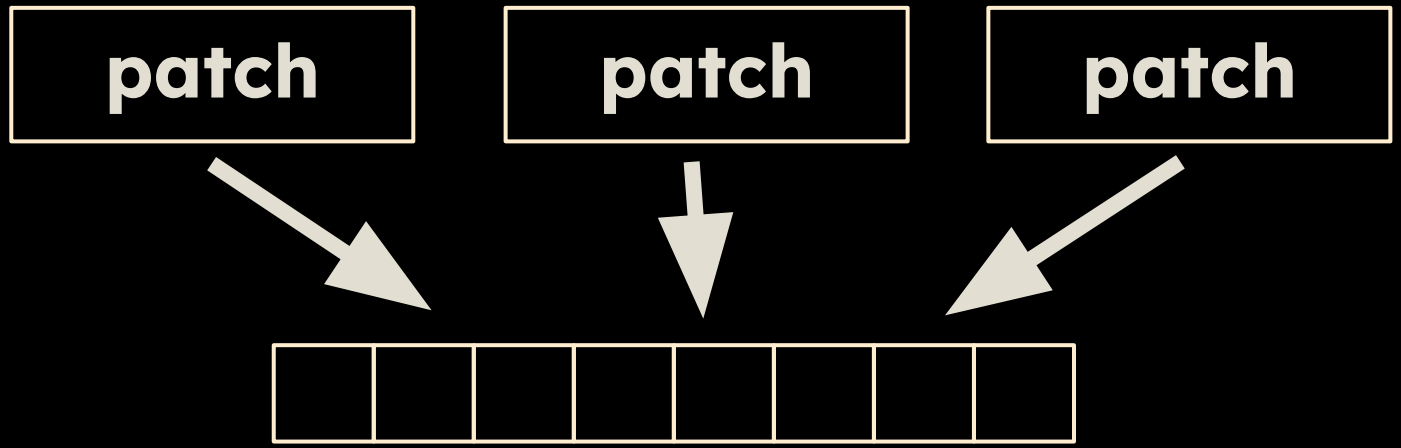
construct a DAG

the edsl

```
self.signal3.write(input_slice[i]);

let port4_0: Window<f32> = Window::new(&self.signal3, Range(0, 0));
pass_thru(&port4_0, &mut self.signal4);

let port2_0: Window<f32> = Window::new(&self.signal3, Range(0, 0));
let port2_1: Window<f32> = Window::new(&self.signal4, Range(0, 0));
add(&port2_0, &port2_1, &mut self.signal2);

let port1_0: Window<f32> = Window::new(&self.signal2, Range(-2, 0));
sum_filter(&port1_0, &mut self.signal1);

let port5_0: Window<f32> = Window::new(&self.signal2, Range(-6, 0));
sum_filter(&port5_0, &mut self.signal5);

let port0_0: Window<f32> = Window::new(&self.signal1, Range(0, 0));
let port0_1: Window<f32> = Window::new(&self.signal5, Range(0, 0));
add(&port0_0, &port0_1, &mut self.signal0);
```

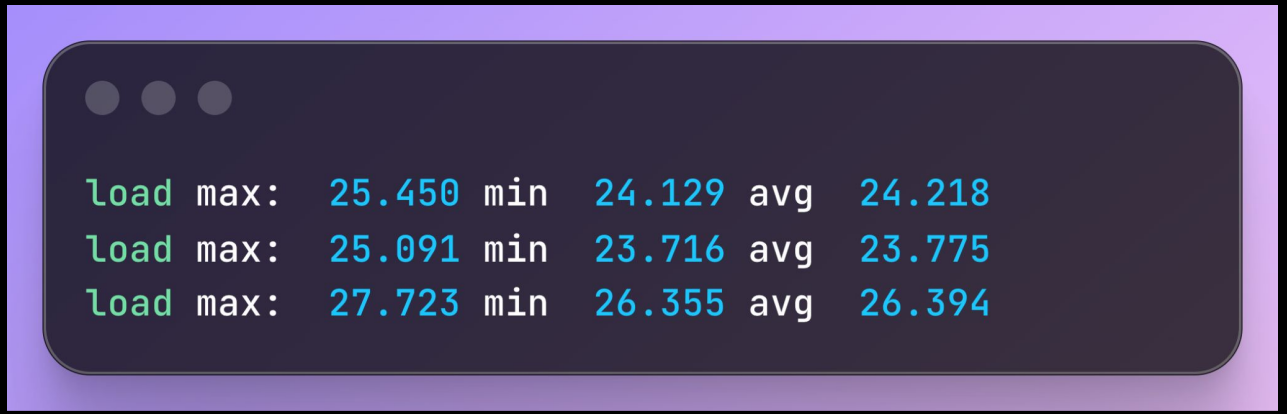
generate rust

the edsl



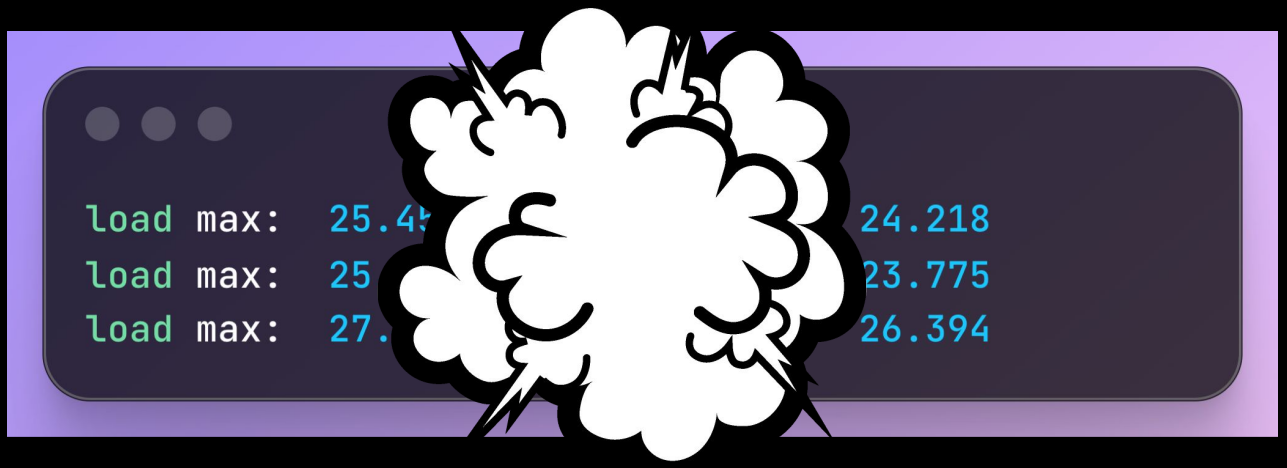
i did share memory

the edsl



faster?

the edsl



i did not make it faster

you cannot outthink the rust inliner

end

a stompbox in rust

Greg Travis
greg.m.travis@gmail.com
https://gregorytravis.me/?1
discord: @mito3705
https://github.com/GregoryTravis

why rust?

cross-compilation is hard