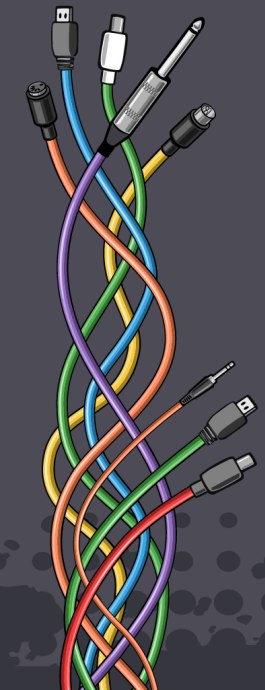


# REAL-TIME AUDIO IN PYTHON

*INTRODUCING THE ASMU PACKAGE*

**FELIX HUBER**

ADC25 Bristol

# Why another audio package?

**Disclaimer**

I am not a professional audio developer, and Python has limitations when it comes to real-time audio processing or applications.

There are several high level real-time audio Python packages e.g.

- **popsicle** Python JUCE bindings (l.c. 2024)
- **pyo** Audio backend in C (l.c. 2023)
- ...

and packages for interfacing hardware (audio drivers) e.g.

- **soundcard** Interface to pulseaudio/coreaudio/WASAPI in pure Python
- **sounddevice** Bindings for PortAudio
- **pyaudio** Bindings for PortAudio (l.c. 2023)
- ...

# Introducing asmu

- High level package with modular structure
- User does not touch the audio data itself
- Pure Python back- and frontend

pypi.org/project/asmu/

## Features

+ Many blocks implemented
+ Calibration handling
+ Documentation available
+ Compact codebase
+ Easy to pick up
+ Platform independent

## Problems

- No real-time guarantee, OS interrupting interpreter
- Careful development required

# Get Started

## User

1. Installation

```
$ pip install asmu
```

2. Scan devices

```
>>> import asmu
>>> asmu.query_devices()
   0 Microsoft Sound Mapper - Input, MME (2 in, 0 out)
>  1 Microphone Array Intel® Smart , MME (6 in, 0 out)
   2 Microsoft Sound Mapper - Output, MME (0 in, 2 out)
<  3 Speakers (Realtek(R) Audio), MME (0 in, 2 out)
```

3. Study documentation at
   `https://felhub.gitlab.io/asmu/`

## Developer

1. Installation

```
$ git clone git@gitlab.com:felhub/asmu.git
$ cd asmu
$ pip install -e .
```

# Hello World

The “Hello World” of audio software - play a sine wave.

```
1 """hello_world.py"""
2 import asmu
3 import time
```

# Hello World

The “Hello World” of audio software - play a sine wave.

```python
"""hello_world.py"""
import asmu
import time
interface = asmu.Interface()
sine = asmu.generator.Sine(interface, 250)
```

# Hello World

The “Hello World” of audio software - play a sine wave.

```python
"""hello_world.py"""
import asmu
import time
interface = asmu.Interface()
sine = asmu.generator.Sine(interface, 250)
sine.output().connect(interface.ioutput(ch=1))
sine.output().connect(interface.ioutput(ch=2))
```

# Hello World

The “Hello World” of audio software - play a sine wave.

```python
"""hello_world.py"""
import asmu
import time
interface = asmu.Interface()
sine = asmu.generator.Sine(interface, 250)
sine.output().connect(interface.ioutput(ch=1))
sine.output().connect(interface.ioutput(ch=2))
with interface.start():
    time.sleep(5)
```

# Hello World

The “Hello World” of audio software - play a sine wave.

```
1  """hello_world.py"""
2  import asmu
3  import time
4  interface = asmu.Interface()
5  sine = asmu.generator.Sine(interface, 250)
6  sine.output().connect(interface.ioutput(ch=1))
7  sine.output().connect(interface.ioutput(ch=2))
8  with interface.start():
9      time.sleep(5)
```

Run the python script with

```
$ python hello_world.py
```

# Module Structure

## Standard imports

| | |
|---|---|
| `asmu` | Topmodule, holds `ASetup`, `AFile`, and `Interface` classes. |
| `asmu.generator` | Submodule with e.g. `Player`, `Sine`, `Noise`, `Vector` ... |
| `asmu.effect` | Submodule with e.g. `Sum`, `GainRamp`, `Weight`, ... |
| `asmu.analyzer` | Submodule with e.g. `FFT`, `Recorder`, `RMS`, `Oscilloscope` ... |

## Optional imports

| | |
|---|---|
| `asmu.processor` | Holds the processor base classes. |
| `asmu.io` | Holds input and output classes. |
| `asmu.acore` | Holds the audio backend base classes. |
| `asmu.typing` | Holds custom internal types. |
| `asmu.exceptions` | Holds custom exceptions. |

# AP: Acoustic Measurement

Microphone frequency response calibration based on IEC 60268-4. Simultaneous comparison to reference.

- Lab calibration of 50+ microphones
- Automated Python workflow
- Easy database integration

Used for open source microphone testing [1].

[1] F. Huber, F.Toth; MEMS measurement microphone compatible to P48 amplifiers; Hardware X; 2025; https://doi.org/10.1016/j.ohx.2025.e00627

# Code Example

```
1  """simple_fr.py"""
2  import matplotlib.pyplot as plt
3  import asmu
```

# Code Example

```
1  """simple_fr.py"""
2  import matplotlib.pyplot as plt
3  import asmu
4  interface = asmu.Interface(device="ASIO Fireface",
5                             samplerate=192000)
6  noise = asmu.generator.Noise(interface, weight="pink")
7  afft = asmu.analyzer.FFT(interface)
```

# Code Example

```python
"""simple_fr.py"""
import matplotlib.pyplot as plt
import asmu
interface = asmu.Interface(device="ASIO Fireface",
                           samplerate=192000)
noise = asmu.generator.Noise(interface, weight="pink")
afft = asmu.analyzer.FFT(interface)
noise.output().connect(interface.ioutput(ch=1))
interface.iinput(ch=1).connect(afft.input(0))  # ref
interface.iinput(ch=2).connect(afft.input(1))  # dut
```

# Code Example

```python
"""simple_fr.py"""
import matplotlib.pyplot as plt
import asmu
interface = asmu.Interface(device="ASIO Fireface",
                           samplerate=192000)
noise = asmu.generator.Noise(interface, weight="pink")
afft = asmu.analyzer.FFT(interface)
noise.output().connect(interface.ioutput(ch=1))
interface.iinput(ch=1).connect(afft.input(0))  # ref
interface.iinput(ch=2).connect(afft.input(1))  # dut
with interface.start() as stream:
    rfft = afft.get_rfft()
```

# Code Example

```python
"""simple_fr.py"""
import matplotlib.pyplot as plt
import asmu
interface = asmu.Interface(device="ASIO Fireface",
                           samplerate=192000)
noise = asmu.generator.Noise(interface, weight="pink")
afft = asmu.analyzer.FFT(interface)
noise.output().connect(interface.ioutput(ch=1))
interface.iinput(ch=1).connect(afft.input(0))  # ref
interface.iinput(ch=2).connect(afft.input(1))  # dut
with interface.start() as stream:
    rfft = afft.get_rfft()
fig, ax = plt.subplots()
ax.plot(afft.frequencies, rfft[:, 1]/rfft[:, 0])
plt.show()
```

# AP: Musicking Carpet

Sound sculpture and performance “The Musicking Carpet”.

- Multichannel audio installation
- Sensor based live dynamic volume
- Custom pressure sensor

Part of conceptual art diploma thesis [2].

[2] N. A. Köbli; The Musicking Carpet; Academy of Fine Arts Vienna; 2025; `https://abschlussarbeiten.akbild.ac.at/search_view`

# Code Example

```
1 """simple_carpet.py"""
2 import asmu
3 audio_files = ["apple.wav", "bear.wav", "sea.wav"]
```

# Code Example

```
1 """simple_carpet.py"""
2 import asmu
3 audio_files = ["apple.wav", "bear.wav", "sea.wav"]
4 interface = asmu.Interface()
5 sum = asmu.effect.Sum(interface)
6 sum.output().connect(interface.ioutput(ch=1))
```

# Code Example

```python
"""simple_carpet.py"""
import asmu
audio_files = ["apple.wav", "bear.wav", "sea.wav"]
interface = asmu.Interface()
sum = asmu.effect.Sum(interface)
sum.output().connect(interface.ioutput(ch=1))
afiles: List[asmu.AFile] = []
ramps: List[asmu.effect.GainRamp] = []
for i, file in enumerate(audio_files):
```

# Code Example

```python
"""simple_carpet.py"""
import asmu
audio_files = ["apple.wav", "bear.wav", "sea.wav"]
interface = asmu.Interface()
sum = asmu.effect.Sum(interface)
sum.output().connect(interface.ioutput(ch=1))
afiles: List[asmu.AFile] = []
ramps: List[asmu.effect.GainRamp] = []
for i, file in enumerate(audio_files):
    afile = asmu.AFile(interface, mode="r", path=file)
    afiles.append(afile.open())
    player = asmu.generator.Player(interface, afile, loop=True)
    ramp = asmu.effect.GainRamp(interface, 0, 0.03, scale="log")
    ramps.append(ramp)
    player.output().connect(ramp.input())
    ramp.output().connect(sum.input(i))
```

# Code Example cont.

```
17  stream = interface.start()
18  try:
19      while True:
20          for ch, ramp in enumerate(ramps):
21              value = get_sensor(ch)
22              ramp.set_gain(value)
23  except KeyboardInterrupt:
24      pass
```

# Code Example cont.

```
17  stream = interface.start()
18  try:
19      while True:
20          for ch, ramp in enumerate(ramps):
21              value = get_sensor(ch)
22              ramp.set_gain(value)
23  except KeyboardInterrupt:
24      pass
25  finally:
26      stream.stop()
27      stream.close()
28      for afile in afiles:
29          afile.close()
```

# Module Structure

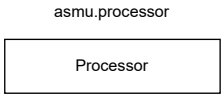
## Standard imports

| | |
|---|---|
| `asmu` | Topmodule, holds `ASetup`, `AFile`, and `Interface` classes. |
| `asmu.generator` | Submodule with e.g. `Player`, `Sine`, `Noise`, `Vector` ... |
| `asmu.effect` | Submodule with e.g. `Sum`, `GainRamp`, `Weight`, ... |
| `asmu.analyzer` | Submodule with e.g. `FFT`, `Recorder`, `RMS`, `Oscilloscope` ... |

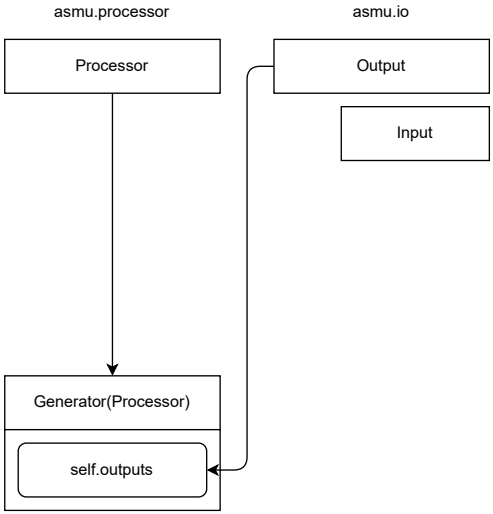
## Optional imports

| | |
|---|---|
| `asmu.processor` | Holds the processor base classes. |
| `asmu.io` | Holds input and output classes. |
| `asmu.acore` | Holds the audio backend base classes. |
| `asmu.typing` | Holds custom internal types. |
| `asmu.exceptions` | Holds custom exceptions. |

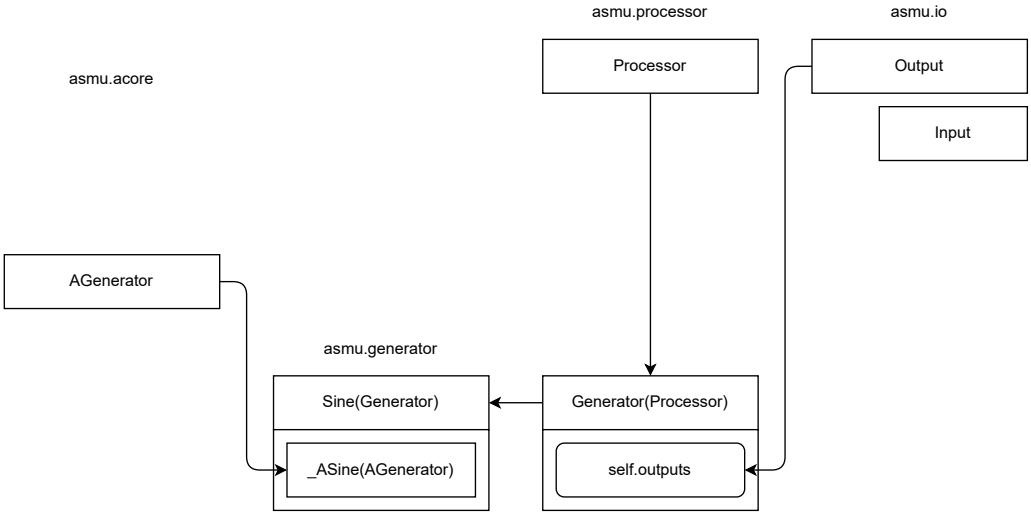
# Class Structure Example



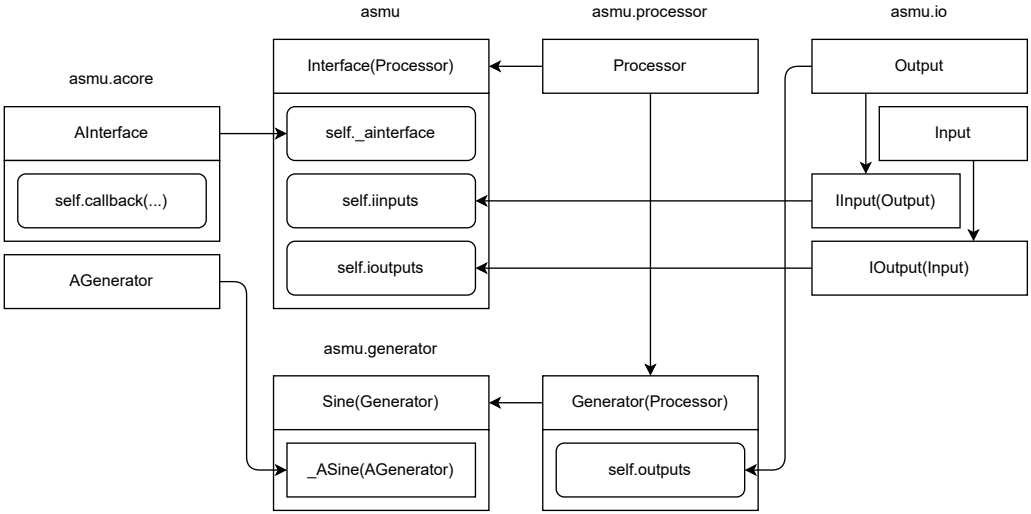
# Class Structure Example



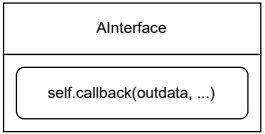
# Class Structure Example



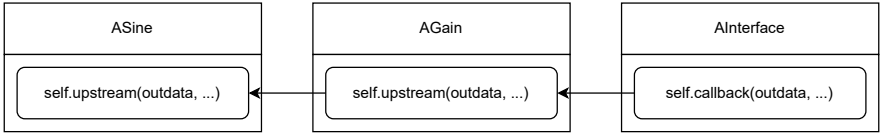
# Class Structure Example



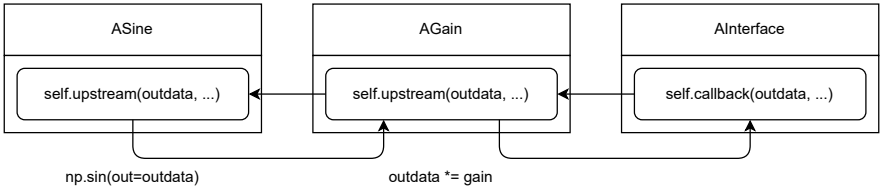
# Audio Backend Principles



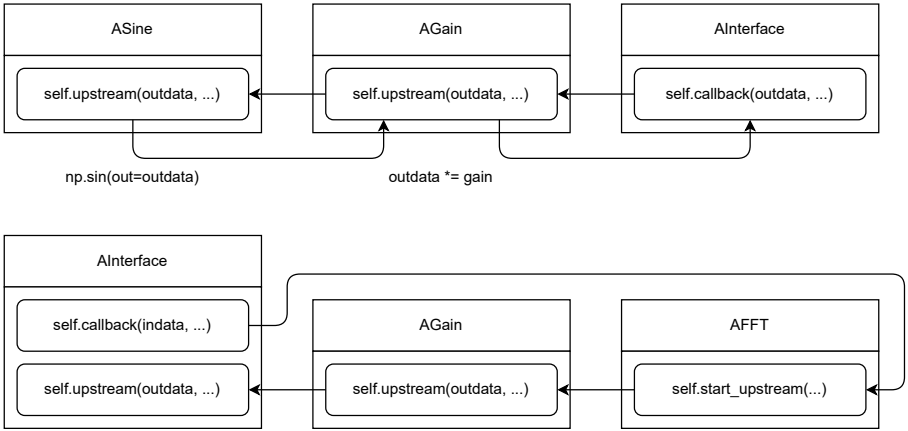
# Audio Backend Principles



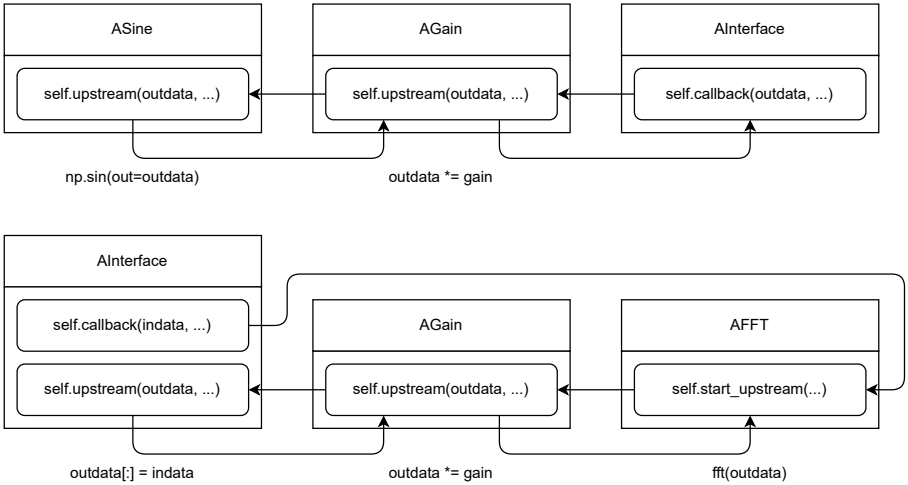
# Audio Backend Principles



# Audio Backend Principles



# Audio Backend Principles



# Profiling

We can mark functions for profiling using the `@profile` decorator.

## line_profiler

```
$ kernprof -l -v ./test_generator_sineburst.py
Line    Time    Line Contents
==============================================================
280             @profile
281             def _mod(self, outdata: "ABlock", ch: int) -> None:
282      1.4      if self._phase < self._maxang:
283     20.9        np.sin(self._omegas+self._phase, out=outdata)
284     14.2        outdata[self._omegas+self._phase >= self._maxang] = 0
```

# Profiling

We can mark functions for profiling using the `@profile` decorator.

**memory_profiler**

```
$ python -m memory_profiler ./test_generator_sineburst.py
Line Mem increment Line Contents
================================================================
280  101.957 MiB   @profile
281                def _mod(self, outdata: "ABlock", ch: int) -> None:
282    0.000 MiB     if self._phase < self._maxang:
283    0.012 MiB       np.sin(self._omegas+self._phase, out=outdata)
284    0.043 MiB       outdata[self._omegas+self._phase >= self._maxang] = 0
```

# Automated Testing

We use `pytest` and GitLab CI for automated integration and unit testing.

- Integration tests of small subgroups e.g. test `FFT` with `Sine`
- Benchmarking of callback function via `pytest-benchmarking`
- Coverage tests via `pytest-cov`

NEW Unit tests with mocking via `pytest-mock`

SOON Hardware testing via local GitLab runner

```
$ pip install asmu
```

```
$ pip install asmu
```

pypi.org/project/asmu/

```
$ pip install asmu
```

pypi.org/project/asmu/

**Thanks to all asmu dependencies <3**

numpy, sounddevice (PortAudio), soundfile (libsndfile), pyFFTW (FFTW)

```
$ pip install asmu
```

pypi.org/project/asmu/

## Thanks to all asmu dependencies <3

numpy, sounddevice (PortAudio), soundfile (libsndfile), pyFFTW (FFTW)

**Felix Huber**
felix.huber@tuwien.ac.at
Technische Universität Wien
E325-03 Sensor and Actuator Technology