MUSIC 420B Project
Implementing “FAUST to Audio Unit”

Reza Payami

1. INTRODUCTION

A Faust program describes a signal processor, a pure computation that maps input signals to output signals. It says nothing about audio drivers or GUI toolkits. This missing information is provided by architecture files. An architecture file describes how to relate a Faust program to the external world, in particular the audio drivers and the user interface to be used. This approach allows a single Faust program to be easily deployed to a large variety of audio standards. The goal of this project is to implement the “Audio Unit” architecture file in order to automatically generate Audio Units based on the given Faust code.

2. ROADMAP

Accomplished Activities

- Read FAUST manual, tutorials and examples, and the architecture file pattern
- Studied VST SDK, the existing VST architecture file, and the generated VST source code
- Studied Audio Unit SDK, some existing examples, and implemented some Audio Units
- Developed the initial Xcode project containing the architecture file and other required classes
- Specified and implemented the initial generalization to implement the architecture file as a wrapper Audio Unit to call FAUST classes
- Used different “mydsp.cpp” files compiled by FAUST, in the Xcode project, to generate and test some initial Audio Units
- Added the multi-channel support
- Implemented the required commands and bash scripts to provide **faust2au (**for simplicity aueffect and au are used interchangeably) to automatically generate the Audio Units
- Measured the latency of the generated Audio Units
- Implemented **faust2ausynth** with polyphony support, by performing all the required activities as described in the above steps
- Generated Audio Units using the existing Faust examples and tested the results

3. GITHUB REPOSITORY

All the files have been committed under the following GitHub address:

https://github.com/rpayami/faust2au

4. CLASS DIAGRAM

The following figure shows the main classes used for implementing faust2au and faust2ausynth. For simplicity, the overridden methods are only shown in the base classes. Among these base classes, AUMonotimbralInstrumentBase and SynthNote are the ones required for implementing an AU synth, and AUEffect base is the corresponding class for writing an AU effect.

"faust -a" command generates "myDSP" class which is included in the source file of FaustAUEffect or FaustAUSynth. These two classes are the main components in the related architecture files, named "au.cpp" and "ausynth.cpp". In general, all the classes with the same color of these two mentioned classes are implemented inside the architecture files.

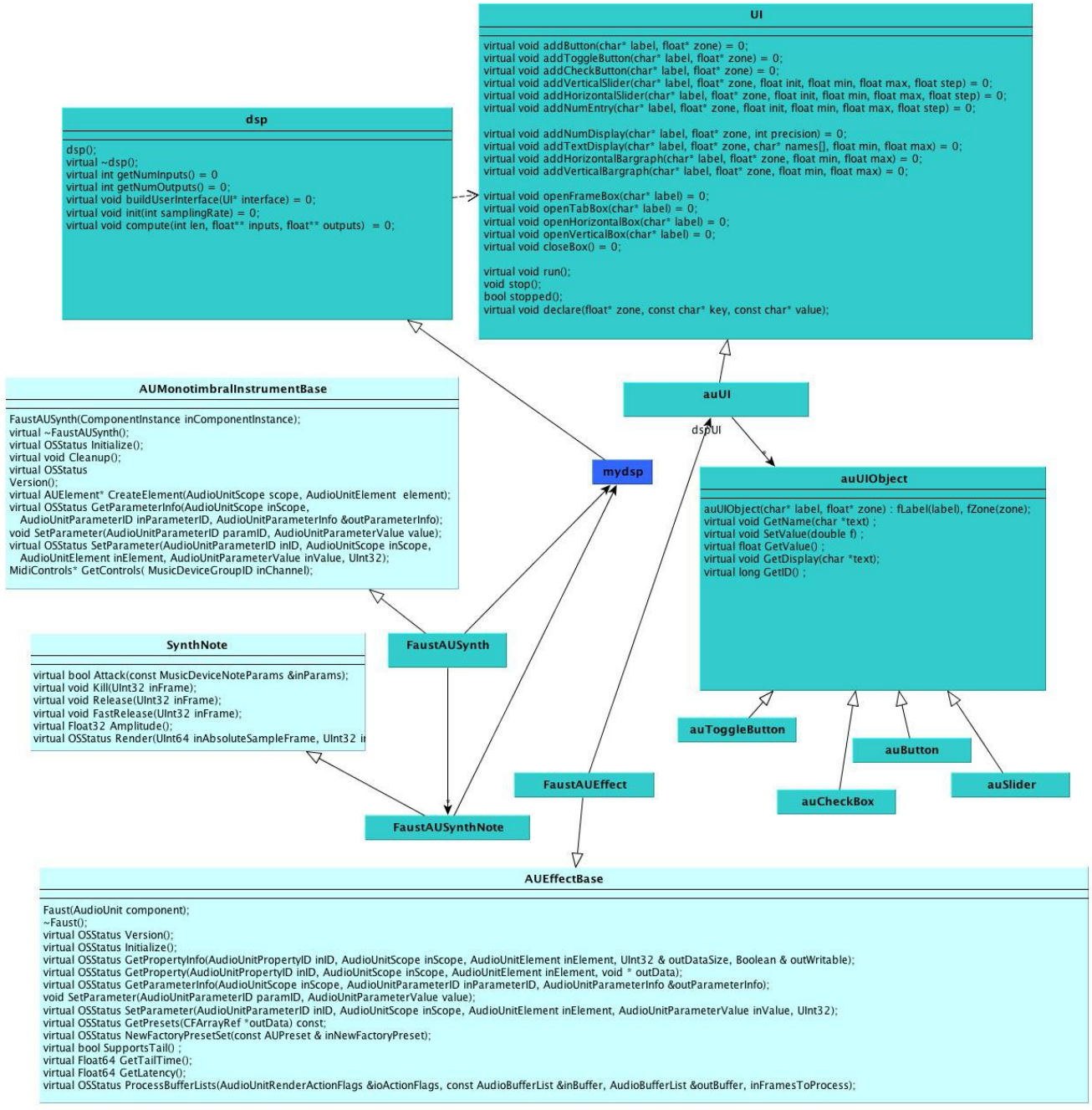

Figure 1 – UML Class diagram for the corresponding faust2au and faust2ausynth classes

5. MEASURING THE LATENCY

To measure the latency, an impulse track was routed to two generated Audio Units, “Feedback Comb Filter with Delay”, and “Zita Reverb”. The Audacity zoomed-out and zoomed-in tracks are shown in the following figures. The zoomed-in figures are the snapshots of the maximum possible Audacity zoom-in, and the x-axis represents time in seconds. Based on the shown units no latency can be observed; as the conclusion the latency should be less than 0.1 millisecond.

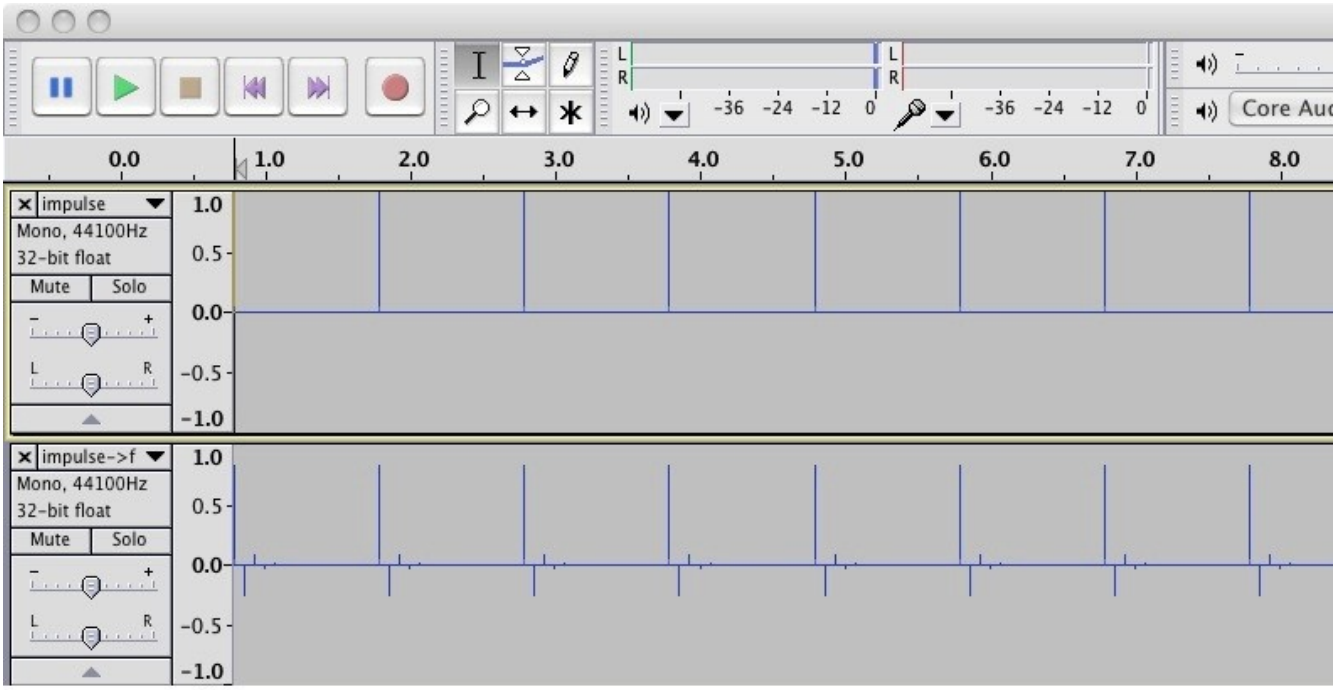

Figure 2 - Zoomed-out snapshot for “Feedback Comb Filter with Delay”

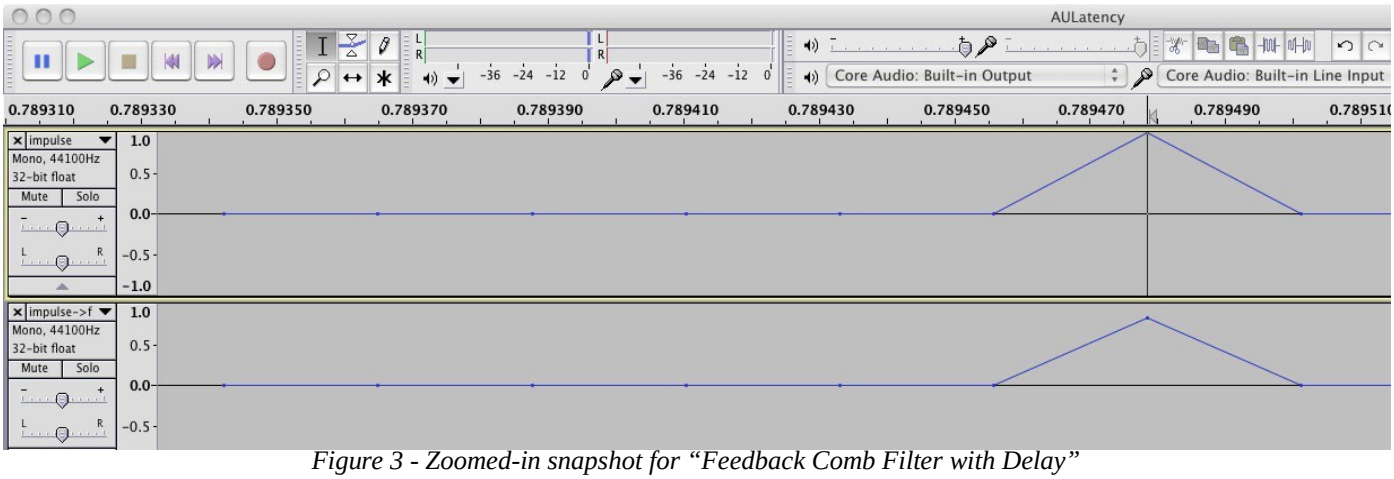

Figure 3 - Zoomed-in snapshot for “Feedback Comb Filter with Delay”

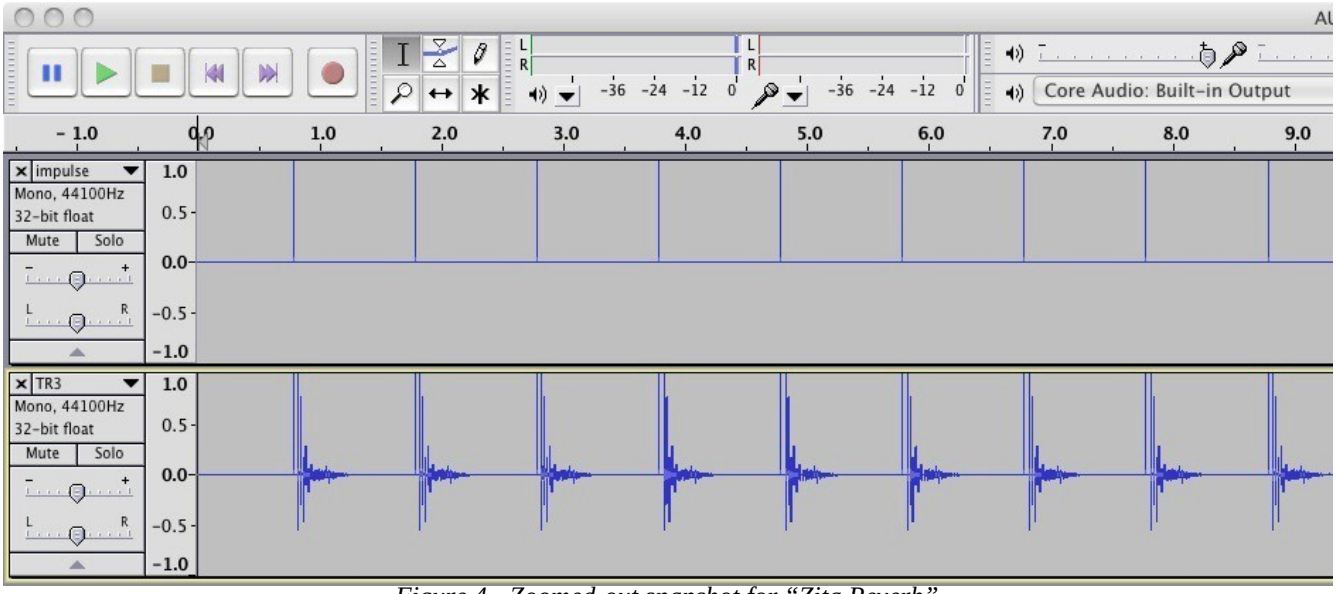

Figure 4 - Zoomed-out snapshot for "Zita Reverb"

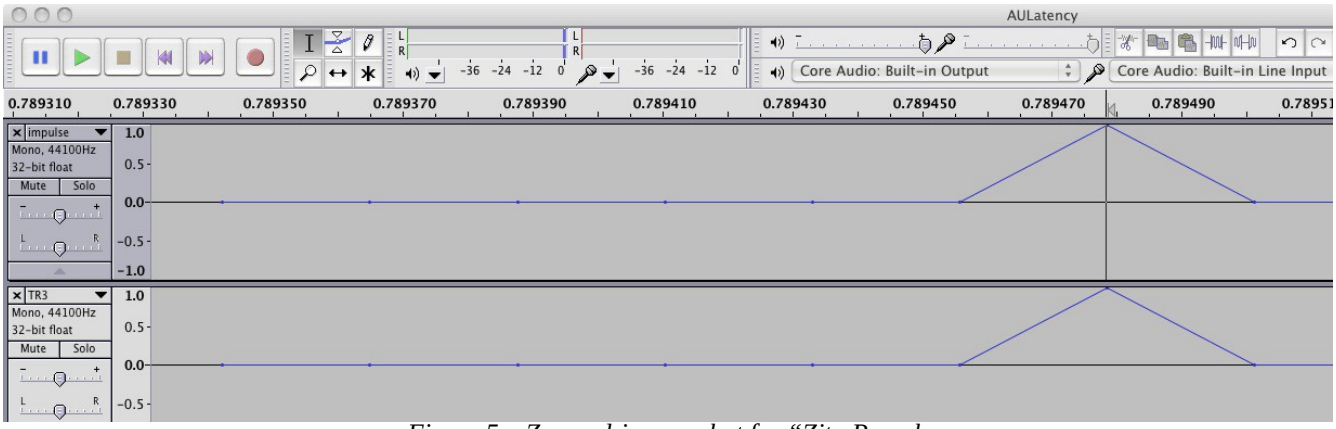

Figure 5 – Zoomed-in snapshot for "Zita Reverb

6. MAIN FILES

- `tools/faust2appls/faust2au`
  `tools/faust2appls/faust2ausynth`
  Bash shell script for generating AU and AUSynth based on a faust dsp file

- `architecture/au.cpp` – AUEffect architecture file
- `architecture/ausynth.cpp` – AUSynth architecture file

- `architecture/AU/Info.plist, architecture/AUSynth/Info.plist`
  The information property list for AUEffect and AUSynth

- `architecture/AU/Info-template.plist, architecture/AUSynth/Info-template.plist`

  The information property list template. A template is used by faust2au and faustausynth to generate AU by replacing some variables based on the command line arguments. The related variables include 'Name', 'Description', 'Subtype', 'Manufacturer', 'BundleId'

- `architecture/AU/FaustAU.xcodeproj, architecture/AUSynth/FaustAUSynth.xcodeproj`

  Xcode projects to be compiled by the bash scripts

- `architecture/AU/FaustAU.exp,architecture/AUSynth/FaustAUSynth.exp`

  Specify the entry points for running the AU executable

- `architecture/AU/Source/AUSource/FaustAUVersion.h`

  `architecture/AUSynth/Source/AUSource/FaustAUSynthVersion.h`

  Specify the subtype, manufacturer and the version of the AudioUnit

- `architecture/AU/Source/AUSource/FaustAUVersion-template.h,`

  `architecture/AUSynth/Source/AUSource/FaustAUSynthVersion-template.h`

  The corresponding templates with 'Subtype' and 'Manufacturer' as the variables.

- `architecture/AU/English.lproj/InfoPlist.strings, architecture/AUSynth/English.lproj/InfoPlist.strings`

  Specify 'BundleName' and 'BundleInfo' of the AUEffect and AUSynth

- `architecture/AU/English.lproj/InfoPlist-template.strings, architecture/AUSynth/English.lproj/InfoPlist-template.strings`

  The corresponding templates with 'BundleName', 'BundleInfo' as the variables.

- `architecture/AU/Source/AUSource/FaustAU.r, and architecture/AUSynth/Source/AUSource/FaustAUSynth.r`

  The resource files describing AUEffect and AUSynth properties

- `architecture/AU/Source/AUSource/FaustAU-template.r, and architecture/AUSynth/Source/AUSource/FaustAUSynth-template.r`

  The corresponding templates with 'Name' and 'Description' as the variables.

7. USAGE

The following describes the installation, command line arguments and the generated output related to both faust2au and faust2ausynth.

Installation

In order to install the latest faust release, and add faust2au update:

- download and extract faust installation archive (http://sourceforge.net/projects/faudiostream/files/faust-0.9.58.zip/download)
- merge the files in the mentioned GitHub repository with the extracted files (replace Makefiles)
- install faust
- Xcode should also be installed on your system (http://developer.apple.com/xcode/)

Running in the Terminal

- with the default arguments:
  faust2au filename (or faust2ausynth filename)
  (e.g. faust2au echo.dsp)
- with some extra arguments:
  faust2au filename manufacturer STYP MANF

  'manufacturer' is the manufacturer of the AU (The default value is 'Grame')

  'STYP' is a four-character subtype (The default is the first four characters of the 'filename')

  'MANF' is a four-character manufacturer (The default is the first four characters of the 'manufacturer')

Output

- ~/Library/Audio/Plug-Ins/Components/'filename'.component

  AU Name will be: 'manufacturer: filename' ('filename' is used as the AU name).

  The names are case-sensitive, e.g. 'faust2au Echo.dsp' results in 'Echo' as the AU name

8. REFERENCES

- Faust Quick Reference, Yann Orlarey, Grame, Centre National de Creation Musicale, April 2006
  http://www.grame.fr/ressources/publications/faust-quick-reference.pdf

- Signal Processing in Faust and PD, Julius Smith, REALSIMPLE Project, CCRMA
  https://ccrma.stanford.edu/~jos/faust/faust.pdf

- Audio Unit Programming Guide, Apple, 2007
  https://developer.apple.com/library/mac/documentation/MusicAudio/Conceptual/AudioUnitProgrammingGuide/AudioUnitProgrammingGuide.pdf

- Steinberg VST, http://www.steinberg.net/en/products/vst.html

- Generating a VST Plugin via Faust, Julius Smith, CCRMA
  https://ccrma.stanford.edu/~jos/faust/Generating_VST_Plugin_Faust.html