Design and Implementation of a Software Sound Synthesizer

Master's Thesis Presentation

Jari Kleimola

August 30, 2005

Outline

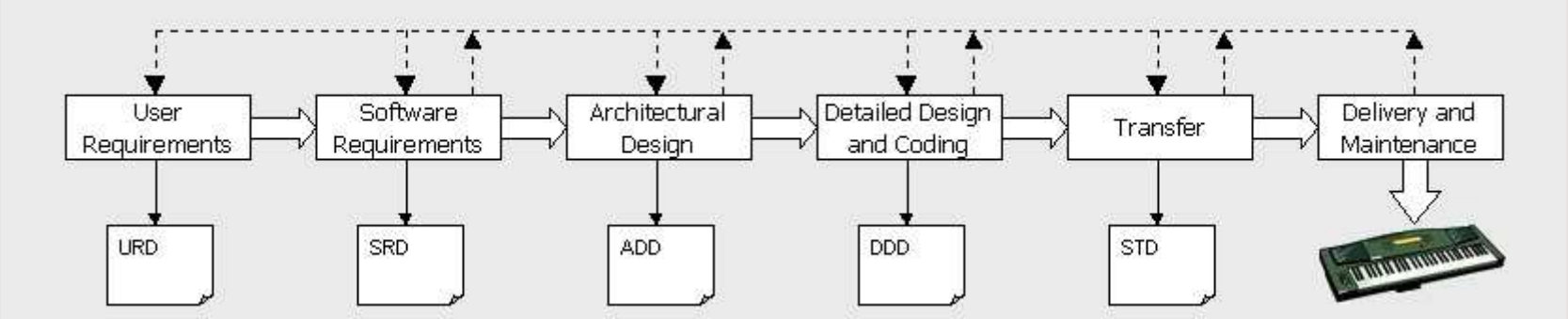
- ✓ Introduction
- ✓ Project Walkthrough
  - ✓ Phases (ESA Model)
  - ✓ Requirements
    - ✓ Synthesizer Architecture
  - ✓ Software
    - ✓ Architectural Design
    - ✓ Implementation
- ✓ Evaluation
  - ✓ Sound
  - ✓ Performance
  - ✓ Code
- ✓ Conclusions and Future Work

Introduction

- ✓ Why yet another software synthesizer ?
  - ✓ What does it take to build one ?
    - ✓ Project (phases, amount of work)
    - ✓ Subsystems
  - ✓ What would my personal dream machine be like ?
    - ✓ Synthesis components
    - ✓ Parameters
    - ✓ MMI
    - ✓ Sound
  - ✓ Create a Software Framework for Synthesis
    - ✓ A place where DSP code can be tested more easily
    - ✓ Explore new methods, learn from existing algorithms
- ✓ The Mission
  - ✓ Fuse elementary synthesis techniques into one composite model
  - ✓ Include some less familiar techniques (Logical AM : XOR, Bézier)

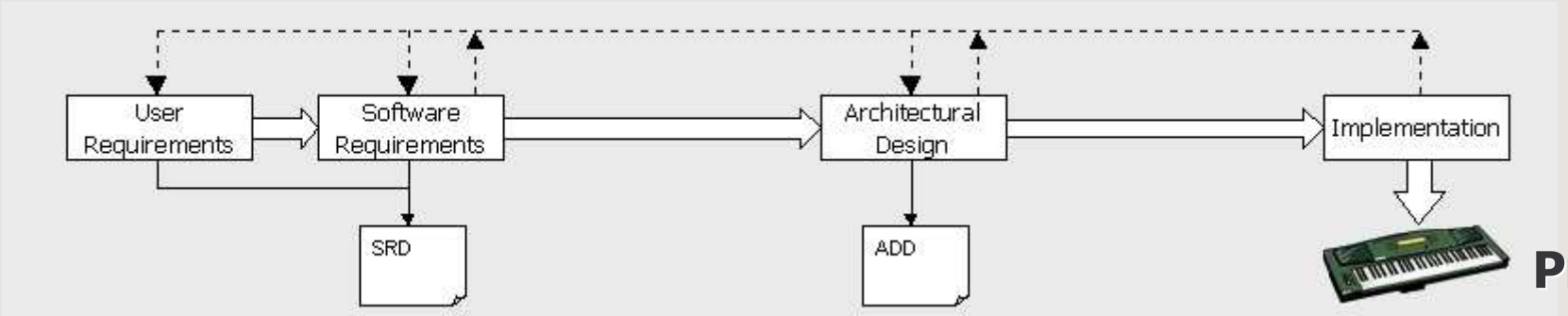
Project Walkthrough

- ESA Software Lifecycle Model (PSS-05-0)



Project Walkthrough

Phases carried out for the prototype



PHUZOR

- Synthesis Architecture
- General Spec
  - Polyphony, Multitimbral...
- Context Diagrams
- Use Cases
  - Load Patch, Play Note...
- Object Models
- Interfaces
- Performance, Safety...

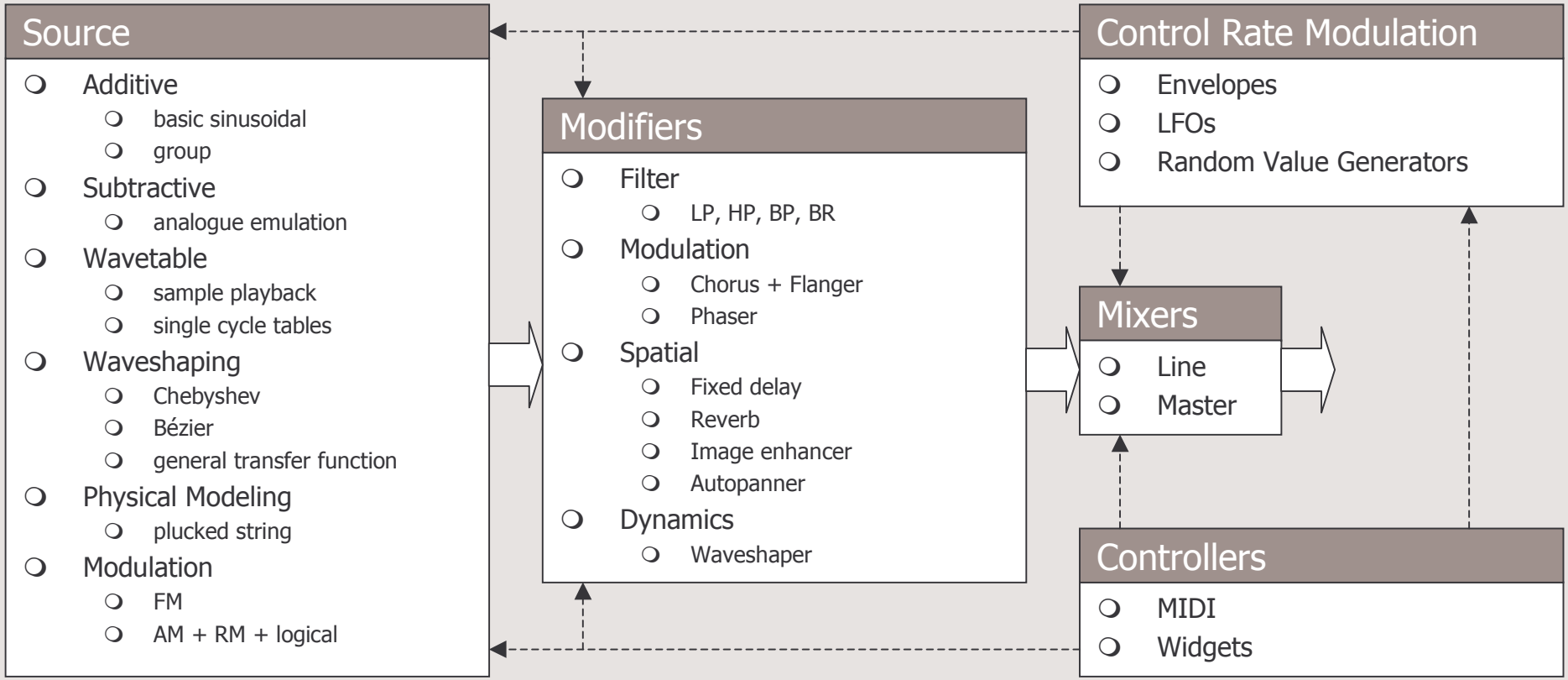
- Subsystems
  - Core, DSP, Midi, MMI...
  - Framework / Phuzor
- Subsystem Interfaces
- Subsystem Internals
  - classes
    - methods
    - members
  - class relationships

- Framework
  - Host + OS wrappers
  - Midi + Audio io
  - Voice allocation
- Phuzor
  - DSP
    - oscillators
    - modifiers
    - modulation
    - containers
  - Patches + parameters
  - MMI etc.

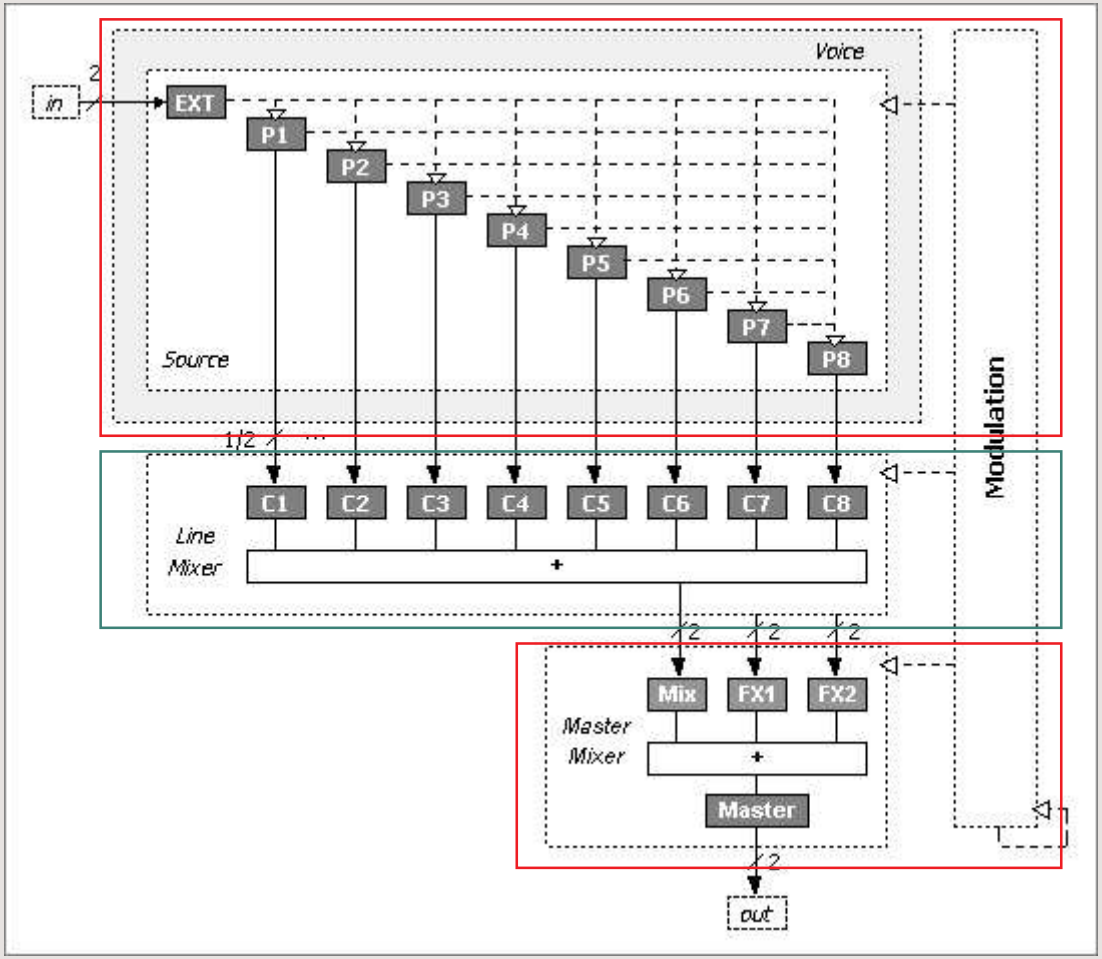
Synthesis Architecture

The Mission

- ✓ Fuse elementary synthesis techniques into one composite model
- ✓ Include some less familiar techniques (Logical AM : XOR, Bézier)



Synthesis Architecture

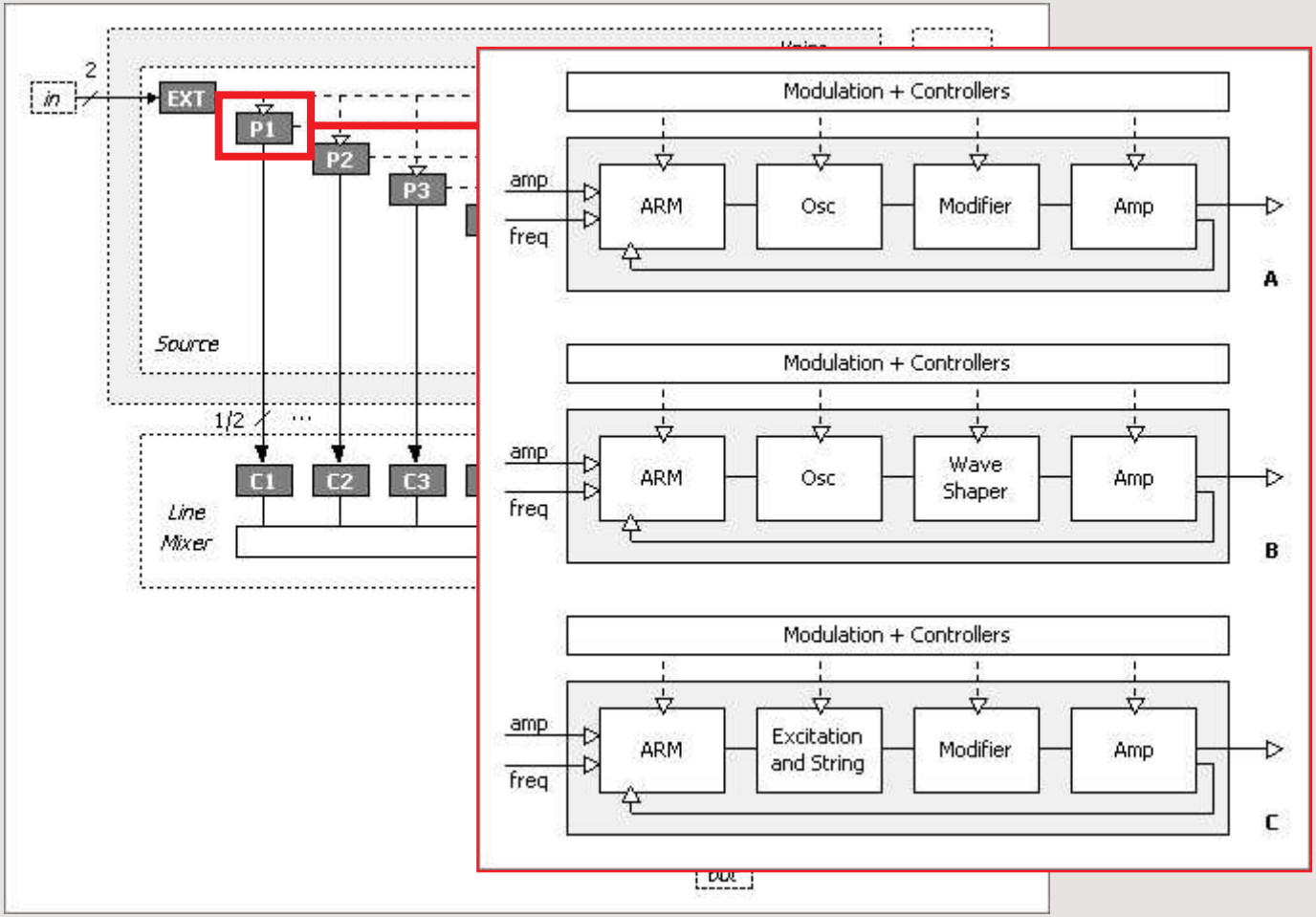


For each voice

$$C1 = \sum_{voice} (P1_{voice})$$

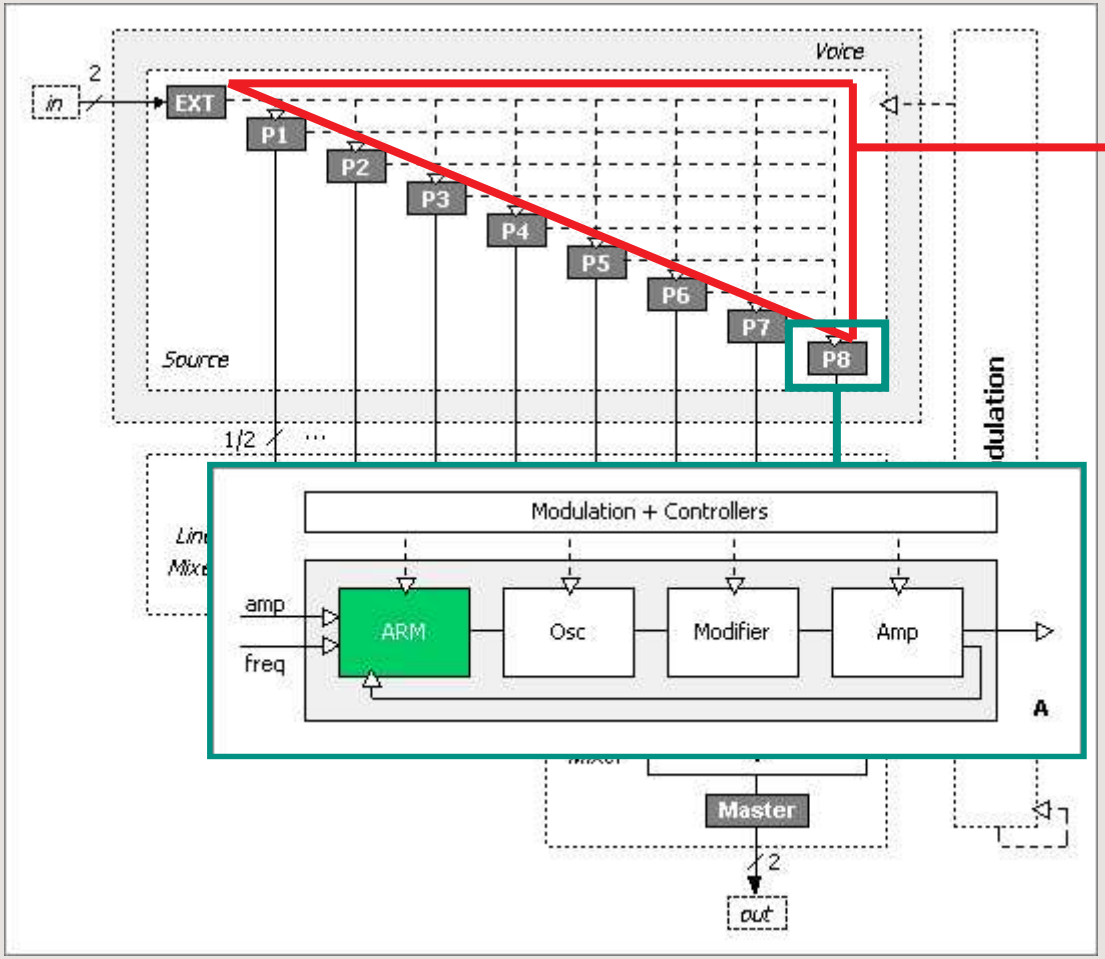
Global for the entire sound

Synthesis Architecture



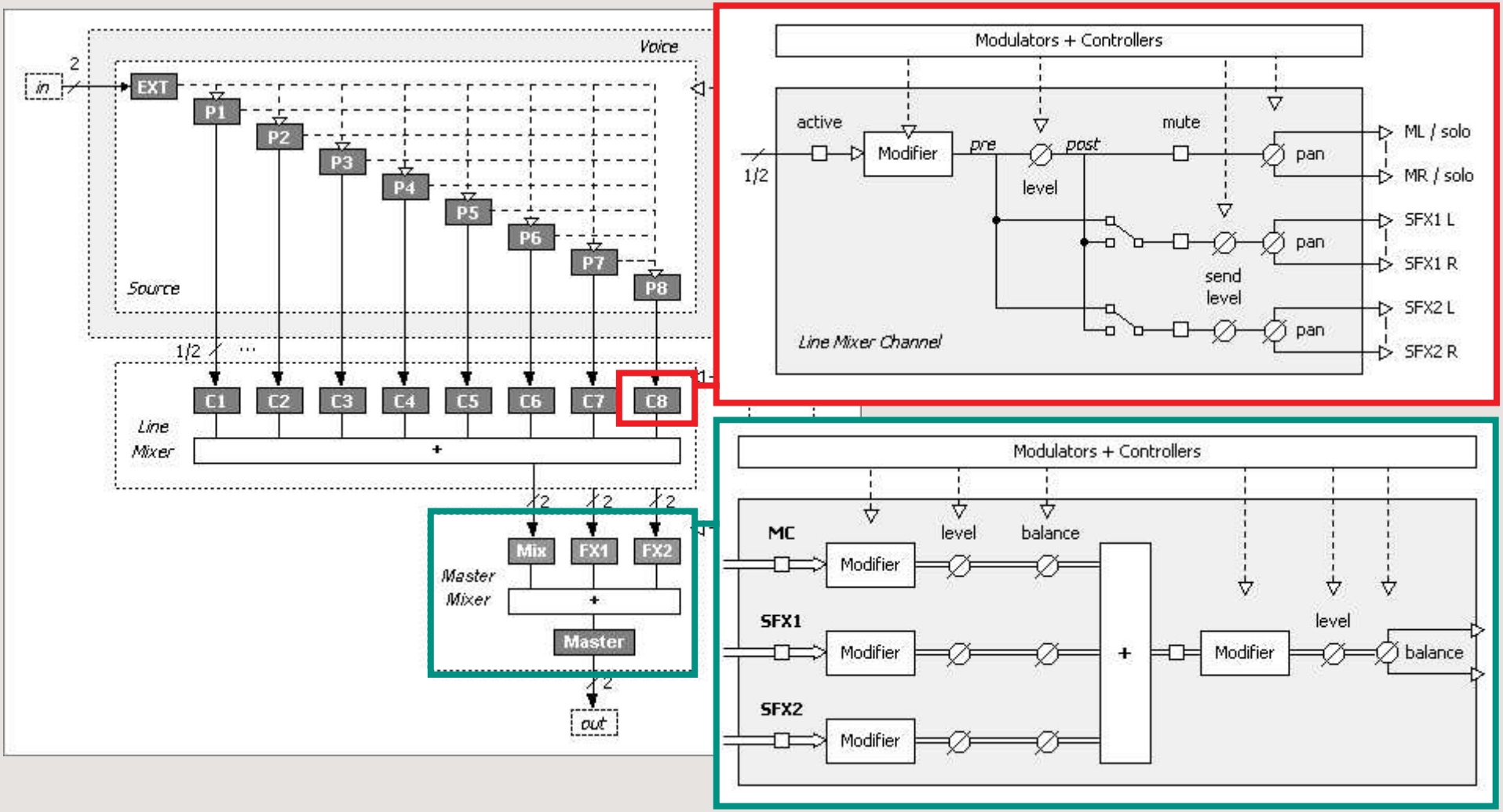
- ✓ Wavetable
  - Additive
  - Subtractive
  - Sample Playback
  - Single Cycle
- ✓ Waveshaper
  - Chebyshev
  - Bézier
  - general
- ✓ Pluck
  - Karplus-Strong
  - Jaffe-Smith
  - Sullivan

Synthesis Architecture

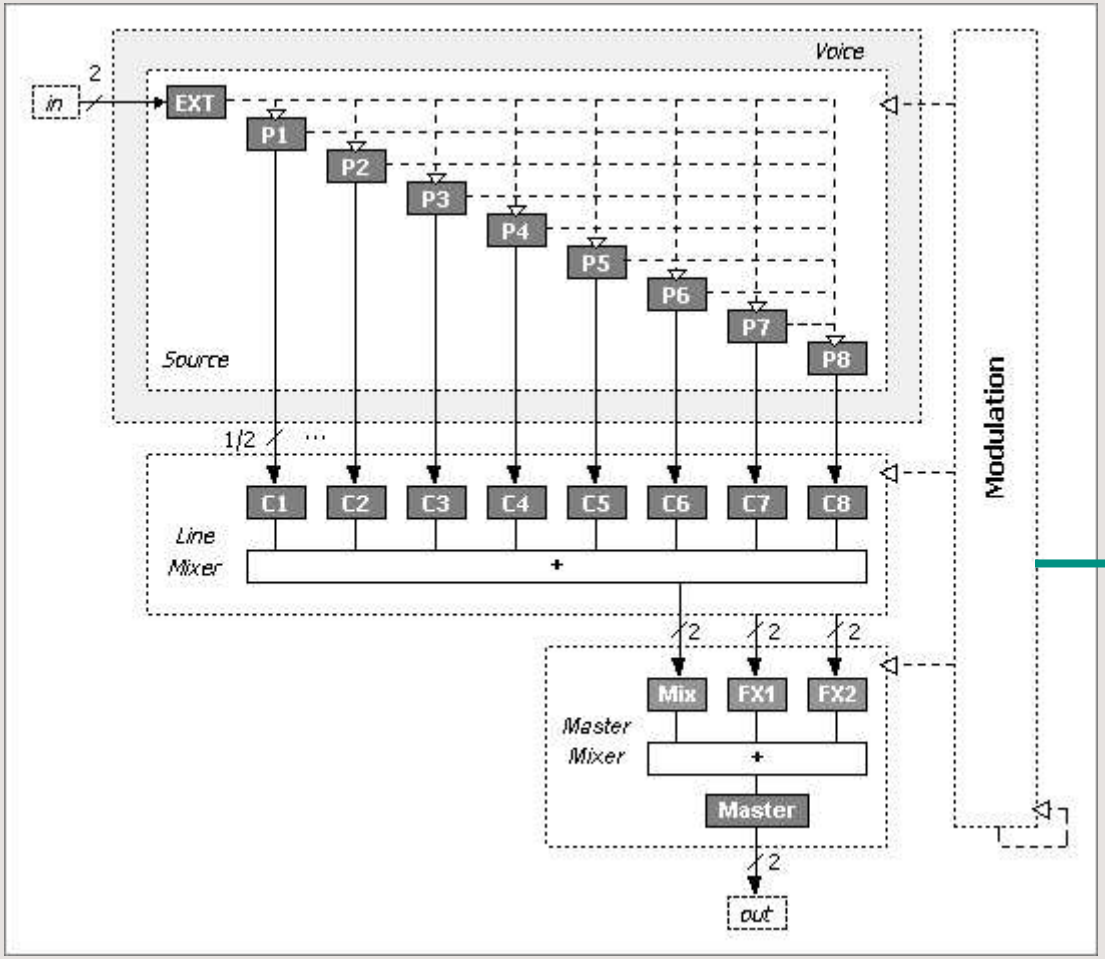


- ✓ Audio Rate Modulation
  - FM
  - AM + RM
  - XOR + OR + AND
  - Sync
  - Mix
  - Pass
- ✓ ARM Matrix
  - source particles
  - destination particles
  - modulation output levels
- ✓ ARM Input Mixer
  - modulation type
  - source particles (a list)
  - input level
  - feedback level

Synthesis Architecture



Synthesis Architecture

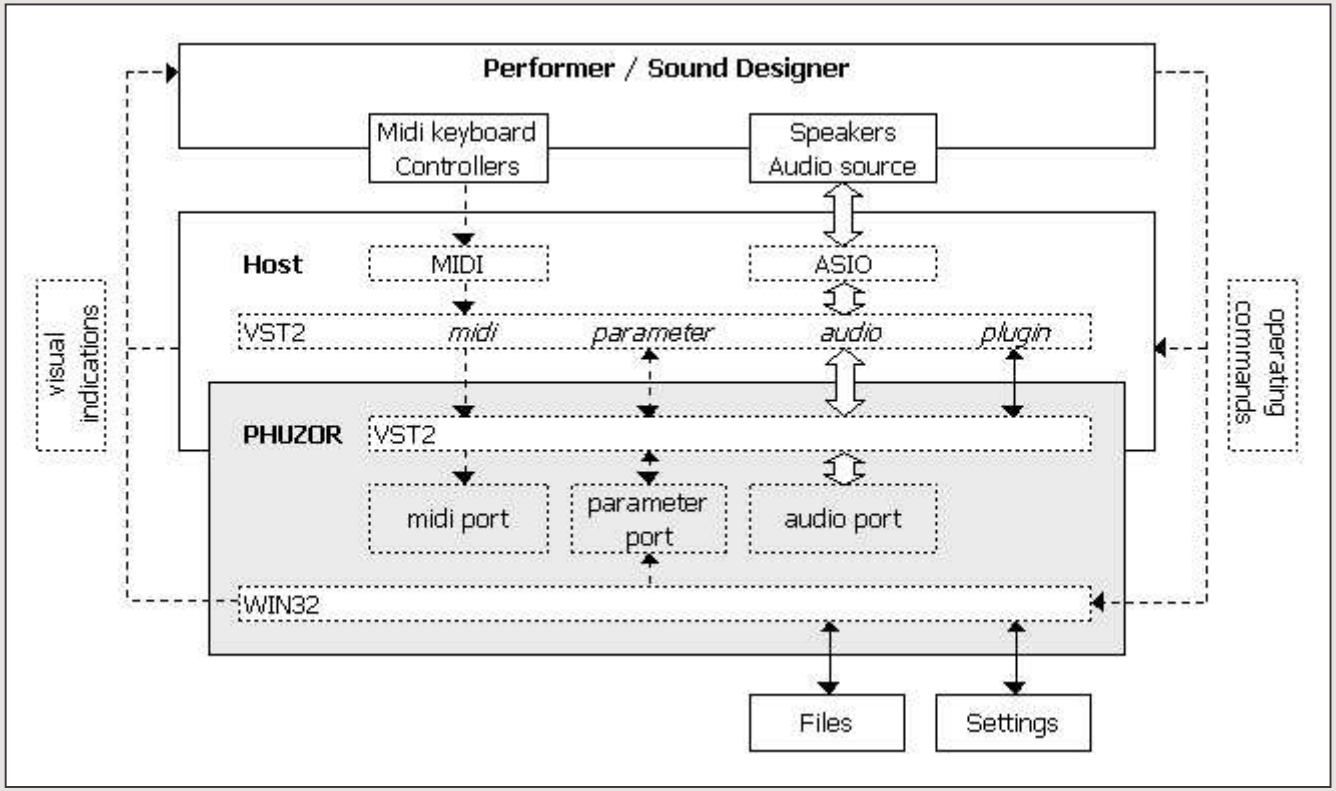


- ✓ Modifier Block
  - topology (serial / parallel)
  - Modifier 1 + 2
    - Filter
    - Spatial
    - Modulation
    - Waveshaper (dynamics)

- ✓ Modulation Matrix
  - Sources
    - EGs
    - LFOs
    - RVGs
    - Controllers
  - Destinations
  - Amount
  - Transformation

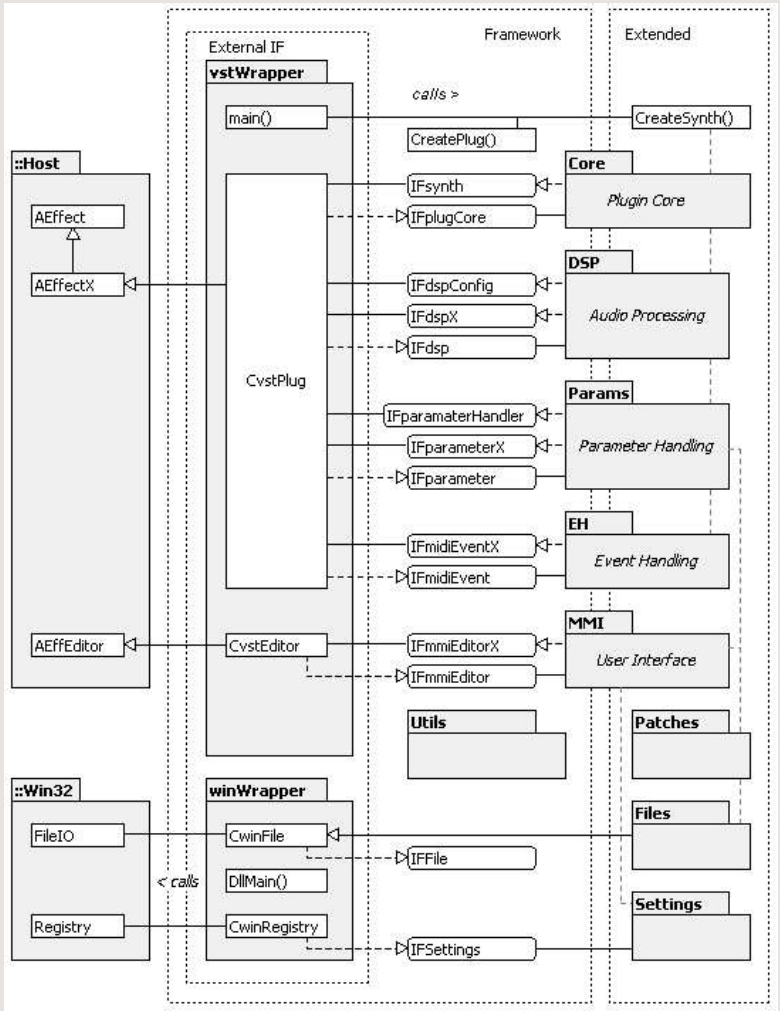
Architectural Design

System Context Diagram



Architectural Design

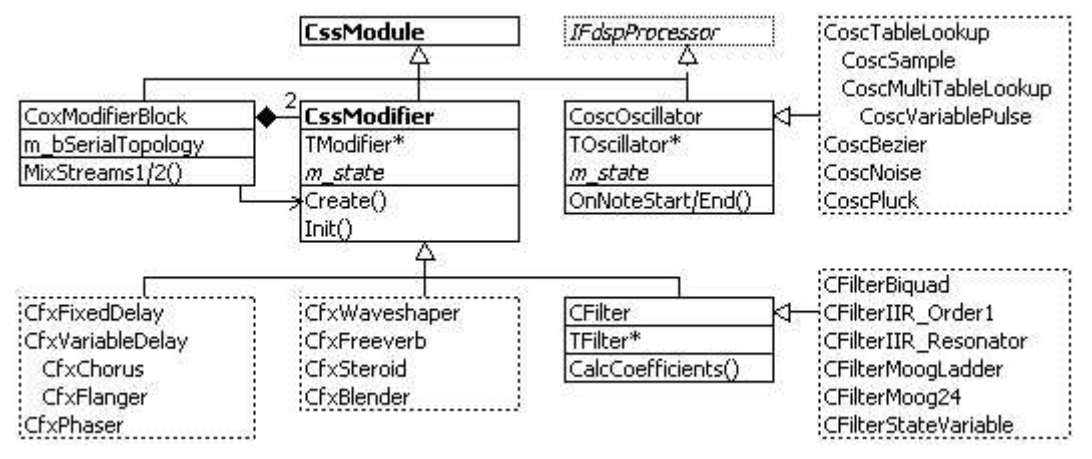
Subsystem Layer



- ✓ Packages
  - Framework
  - Extended (= Phuzor)
- ✓ SubSystems
  - Core
  - DSP
  - Params
  - EH
  - MMI
  - Patches
  - Files
  - Settings
  - Utils
  - vstWrapper (::Host)
  - winWrapper (::Win32)

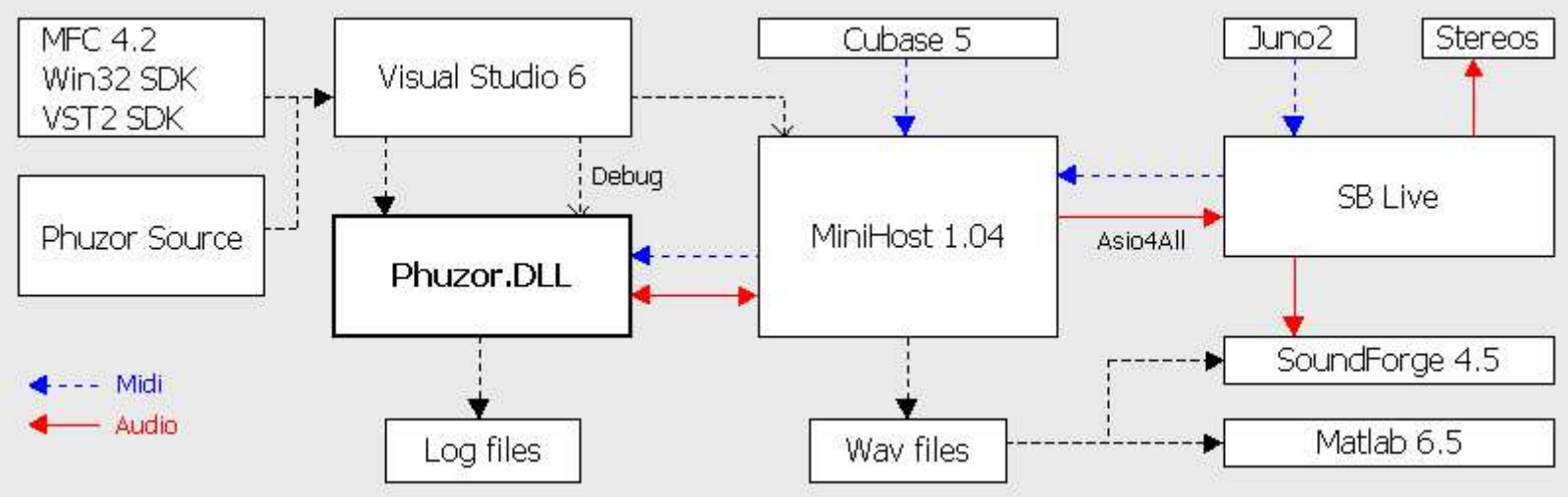
Architectural Design

Example : a subset of DSP classes



Implementation

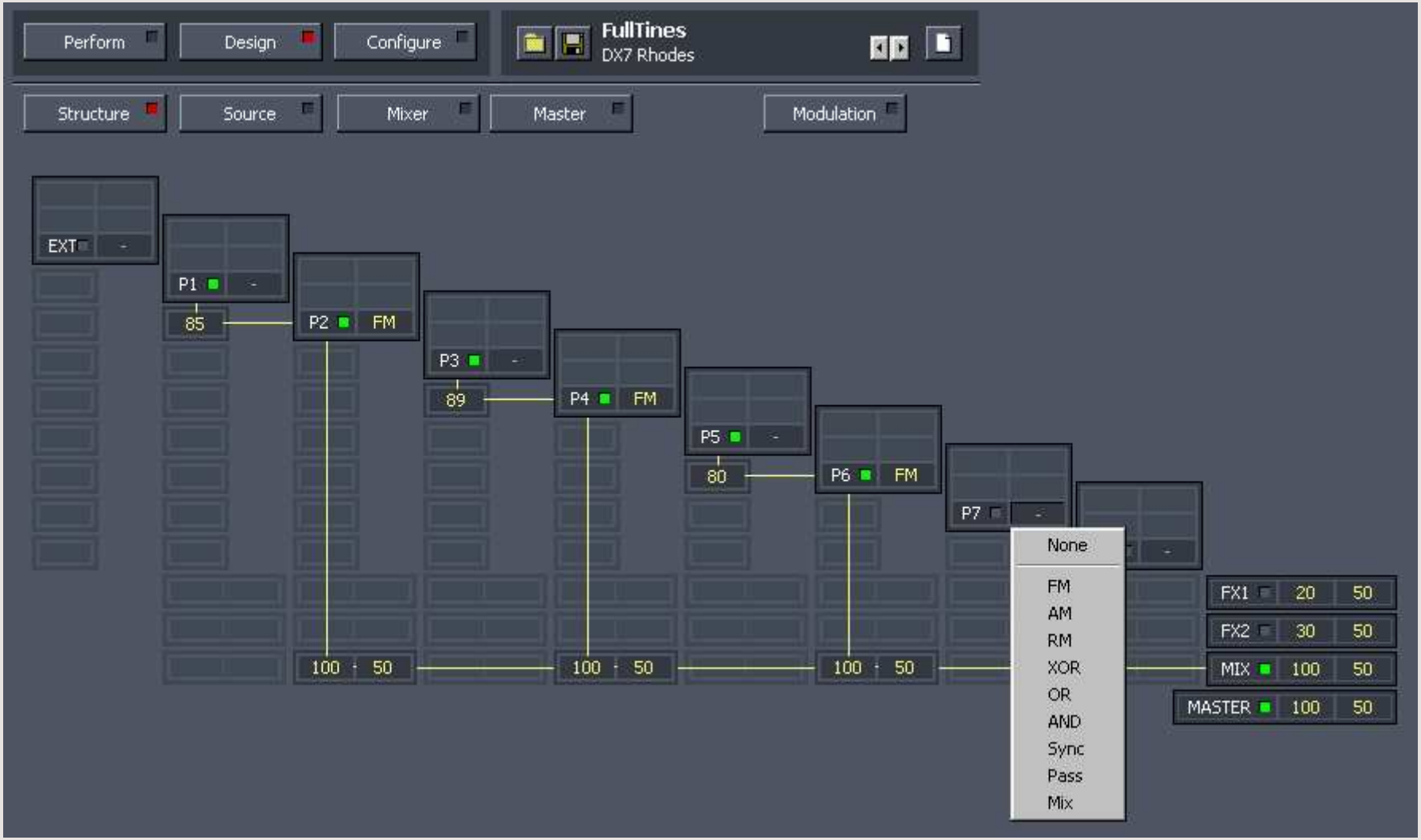
- ✓ Environment



- ✓ Some key algorithms
  - ✓ timeslicing (= concurrent handling of audio and midi)
  - ✓ table lookup oscillators
  - ✓ Bézier + Chebyshev polynomials
  - ✓ audio and control rate modulation
  - ✓ ringbuffer and delay line
  - ✓ filters

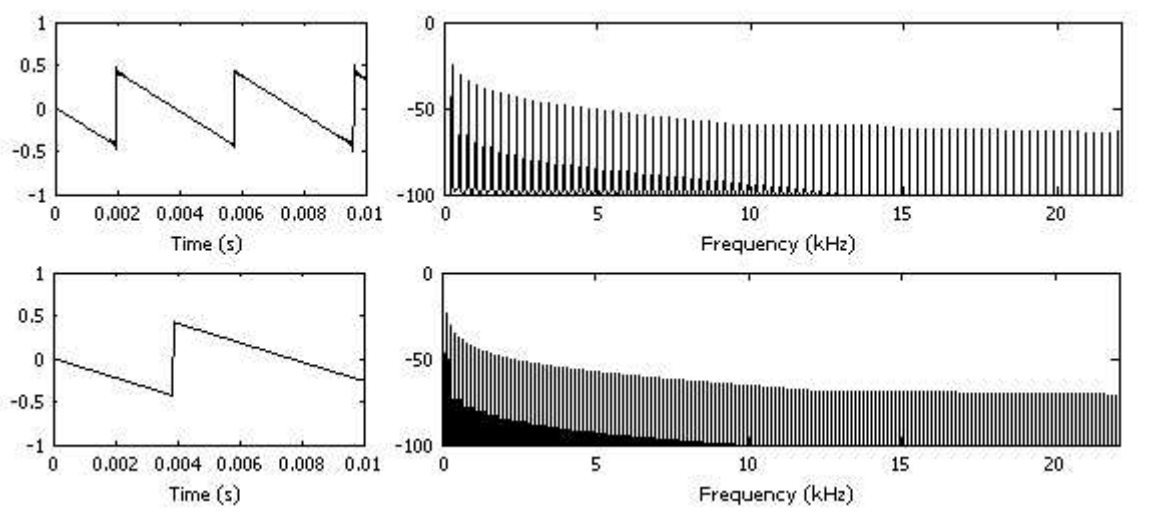
Implementation : MMI example

Combined Line/Master Mixer and Audio Rate Modulation matrix surface



Evaluation -- Sawtooth a

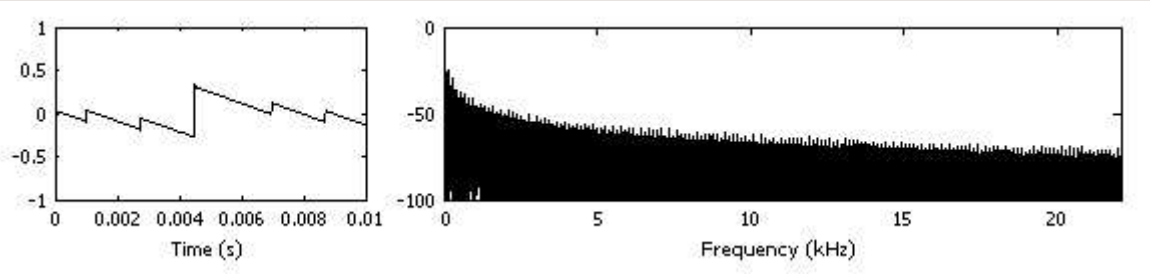
- Stock Saw (Fourier generated wavetables including all harmonics)



C4 261 Hz

C3 130.5 Hz

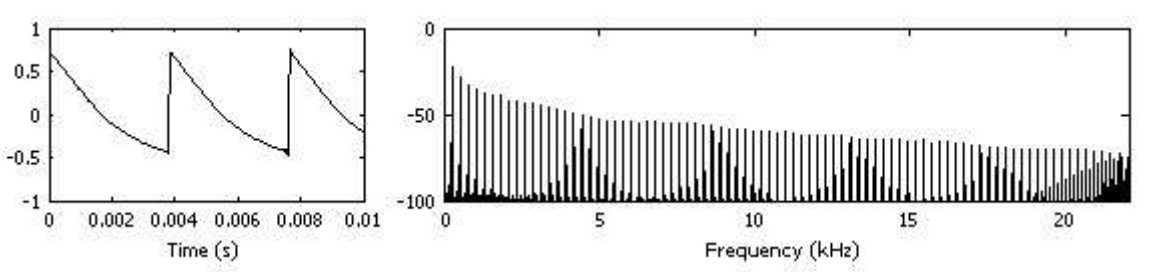
- Super Saw (stock saw + 6 detuned pseudo oscillators)



C3 130.5 Hz

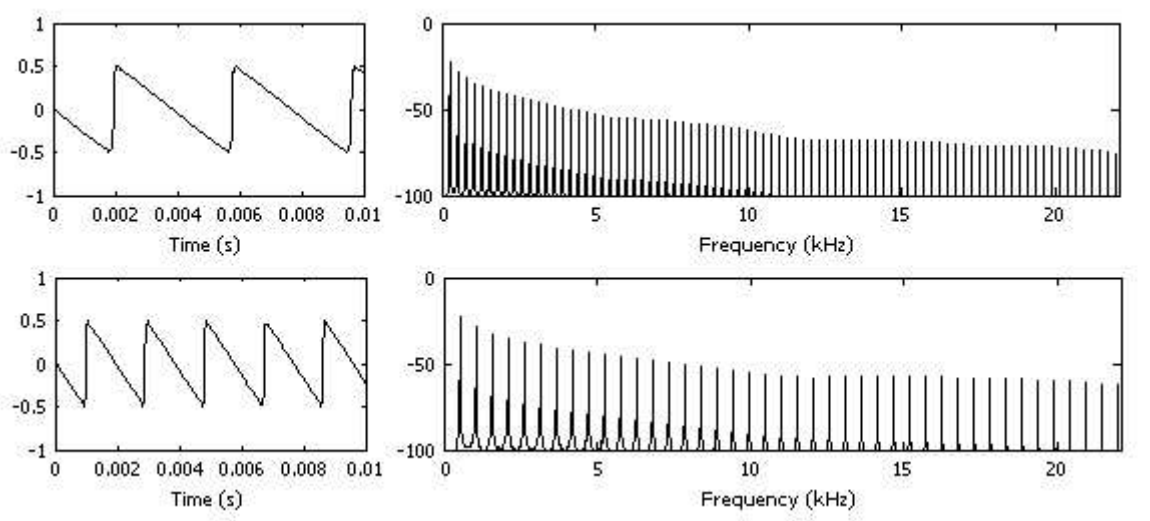
Evaluation -- Sawtooth b

- Sampled Saw (Minimoog, contains aliasing)



C4 261 Hz

- Spectral Saw (Fourier generated wavetable from Moog sample above)

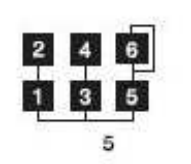


C4 261 Hz

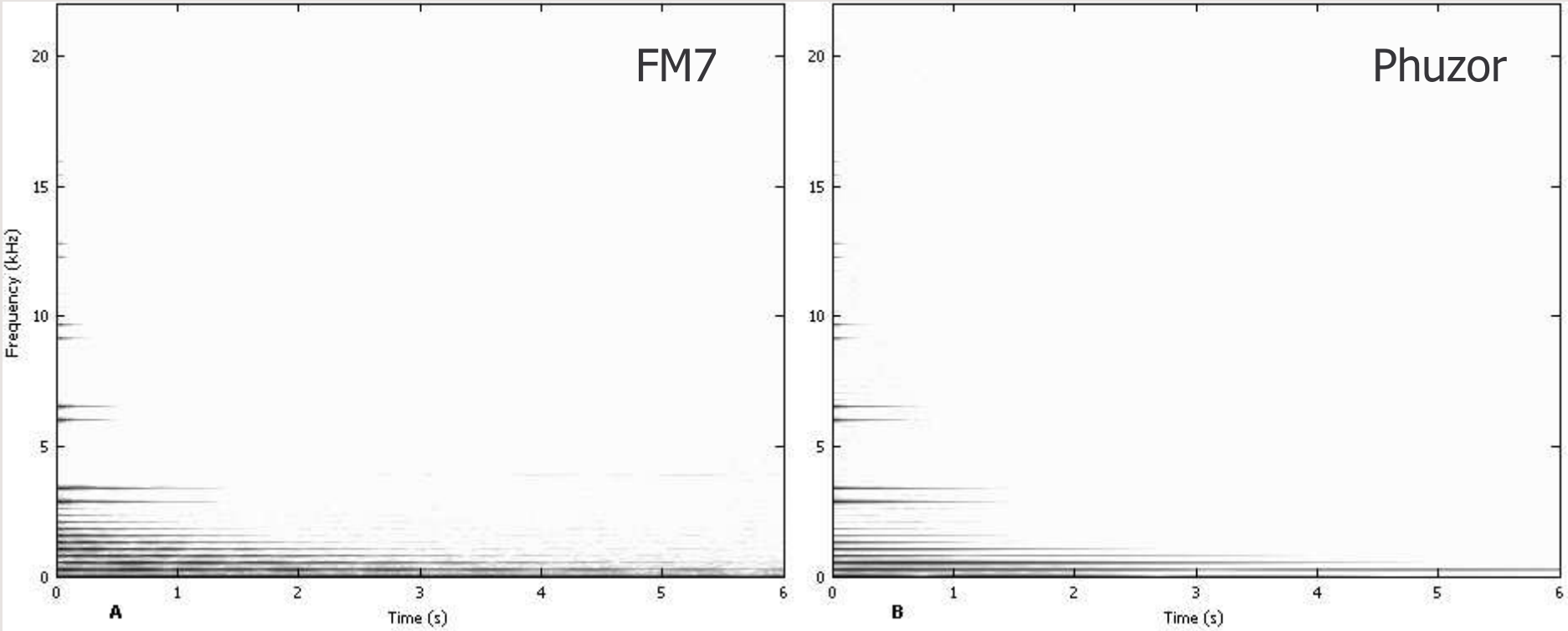
C5 522 Hz

Evaluation -- FullTines

- FM Rhodes Electric Piano



- ✓ sin oscillators
- ✓ op2 / op1 = 12 / 1, others 1 / 1
- ✓ fast decay from full level to zero on op2 EG



Evaluation -- Performance

- ✓ Compiler optimized for speed (no SSE tuning)
- ✓ one voice
- ✓ 44.1 kHz stereo out
- ✓ output buffer size 512 floats per channel ⇨ 5.805 ms per block

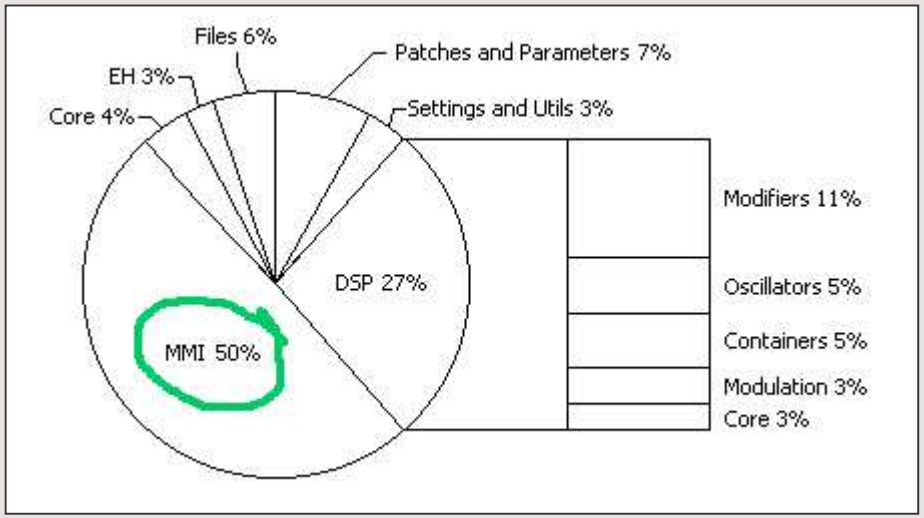
General	**Time (ms)**			
Startup time	10..11 s			⇦ Needs work
MIDI input and MMI update				
MIDI in -> event queue	0.023			
MIDI in -> synthesis start (i.e. latency - synthesis time)	0.056			
Oscillator pitch doubled from MMI -> sound output	7.816			⇦ Windows OS
Synthesis of one output buffer		**CPU (%)**	**Polyphony**	
1 wavetable particle with stock sawtooth wave	0.062	0.7	94	
1 waveshaper particle with Bézier wave	0.120	1.3	48	
1 pluck particle	0.075	0.8	77	
1 wavetable particle with supersaw	0.143	1.3	41	
supersaw synthesized from 7 detuned stock sawtooth waves	0.171	1.6	34	
6 - particle FM patch	0.361	2.6	16	⇦ Needs work
Synthesis of one output buffer using Modifiers				
1 filtered (24 dB LPF) wavetable particle with sawtooth	0.110	1.1	53	
1 wavetable saw particle run through chorus (insert)	0.148	1.4	39	
1 wavetable saw particle run through master reverb (send)	0.296	2.5	20	⇦ Could be better

Evaluation -- Code

- Does size matter ?

	classes	statements	
Entire Project	188	19160	100%
Framework	63	5191	27%
Phuzor	125	13969	73%
csound	0	66968	3.5 times bigger
Cook's STK	98	17511	about same size

- Distribution amongst subsystems



Conclusions

- ✓ Design
  - ✓ Good design makes implementation easier and more robust
  - ✓ ESA design model works, but is too tedious
  - ✓ Object oriented model suits well for synthesis application development
- ✓ Implementation
  - ✓ Real-time requirements make the OO approach unusable sometimes
  - ✓ A lot of MMI code was needed
  - ✓ Debugging is difficult
- ✓ Results
  - ✓ Architecture seems to be flexible, yet fairly intuitive
  - ✓ Sound quality is OK
  - ✓ Don’t know yet if Phuzor has a sound of its own
  - ✓ Final version is realizable

Future Work

- ✓ Release version needs polishing
  - ✓ DSP code optimization (SSE)
  - ✓ MMI
- ✓ DXI plugin port
- ✓ More patches
  - ✓ Standalone tool that converts pre-programmed patches into Phuzor format
  - ✓ Sample library
- ✓ Add more synthesis techniques + modifier algorithms
  - ✓ Physical modeling
  - ✓ Research XOR and Bézier methods
  - ✓ Add hosting
    - ✓ Effects -> Modifier
    - ✓ Instruments -> Particle