MIDI Support for the Subtractive Synth in SuperCollider

Demonstration

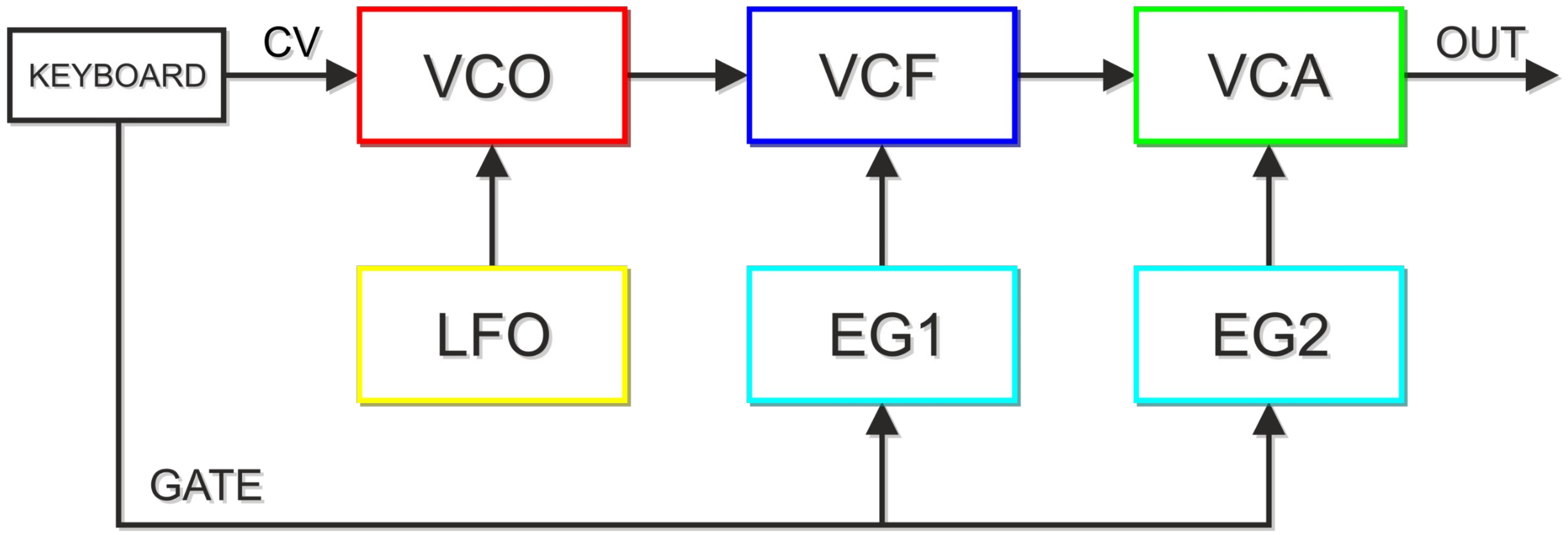
Computer.Music.And.I

Episode 02

Adding MIDI Support to the Subtractive Synth

Computer.Music.And.I

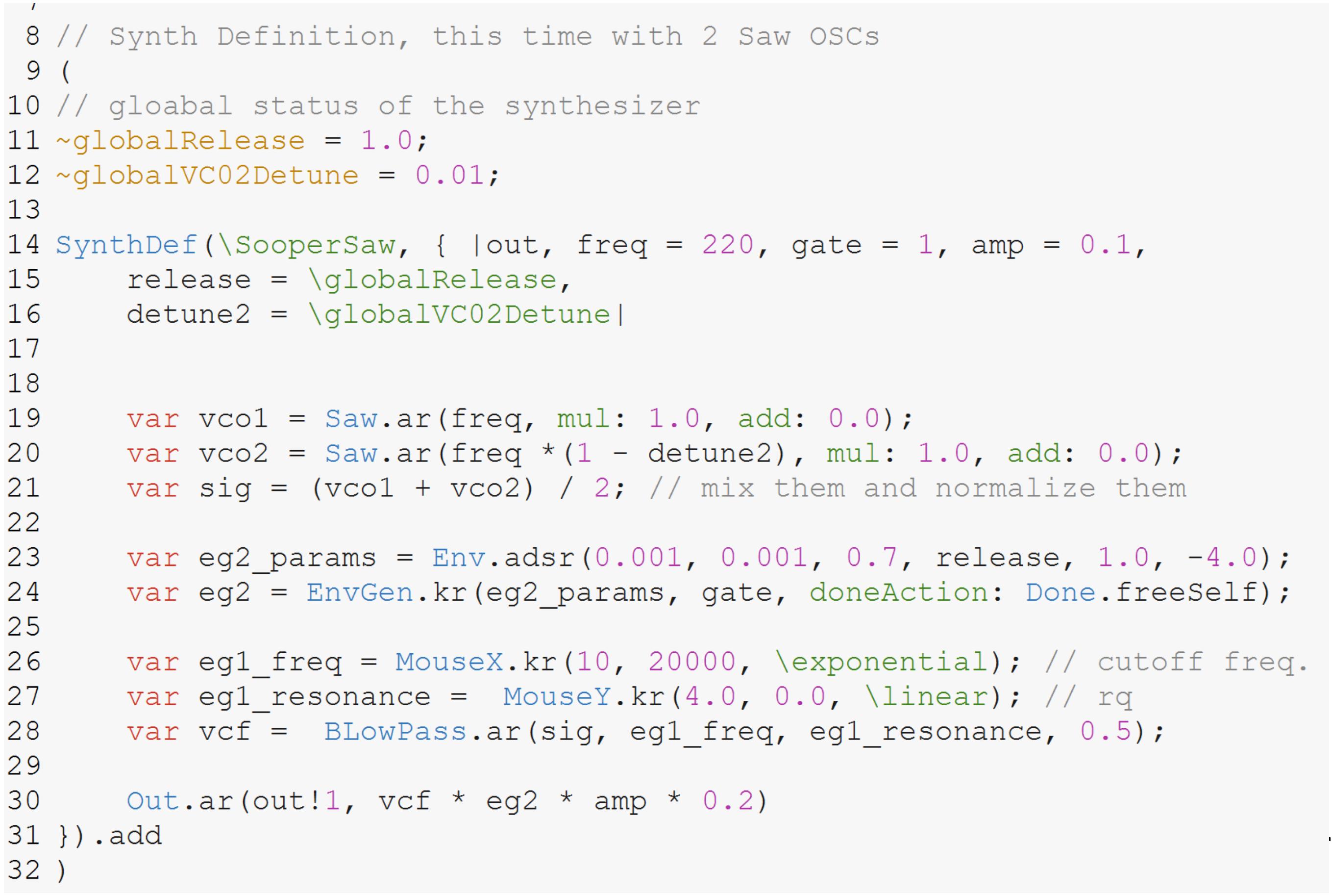
Subtractive Synth: Base



https://en.wikipedia.org/wiki/Synthesizer#Theory

Computer.Music.And.I

Revised version of the subtractive synthesizer

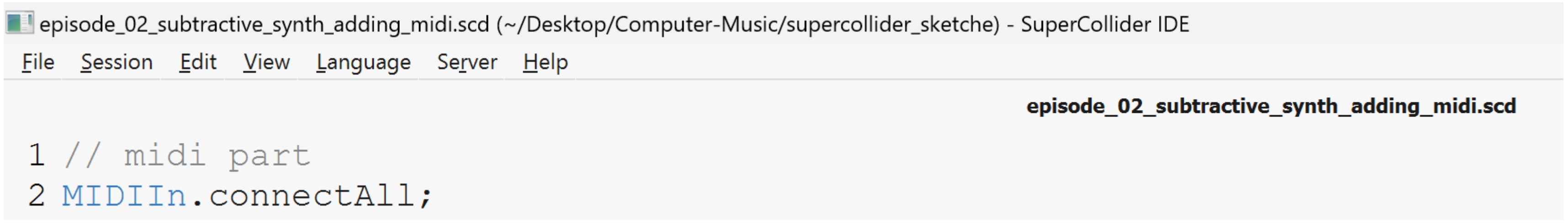


```
// Synth Definition, this time with 2 Saw OSCs
(
// gloabal status of the synthesizer
~globalRelease = 1.0;
~globalVC02Detune = 0.01;

SynthDef(\SooperSaw, { |out, freq = 220, gate = 1, amp = 0.1,
    release = \globalRelease,
    detune2 = \globalVC02Detune|


    var vco1 = Saw.ar(freq, mul: 1.0, add: 0.0);
    var vco2 = Saw.ar(freq *(1 - detune2), mul: 1.0, add: 0.0);
    var sig = (vco1 + vco2) / 2; // mix them and normalize them

    var eg2_params = Env.adsr(0.001, 0.001, 0.7, release, 1.0, -4.0);
    var eg2 = EnvGen.kr(eg2_params, gate, doneAction: Done.freeSelf);

    var eg1_freq = MouseX.kr(10, 20000, \exponential); // cutoff freq.
    var eg1_resonance =  MouseY.kr(4.0, 0.0, \linear); // rq
    var vcf =  BLowPass.ar(sig, eg1_freq, eg1_resonance, 0.5);

    Out.ar(out!1, vcf * eg2 * amp * 0.2)
}).add
)
```

MIDI: Musical Instrument Digital Interface

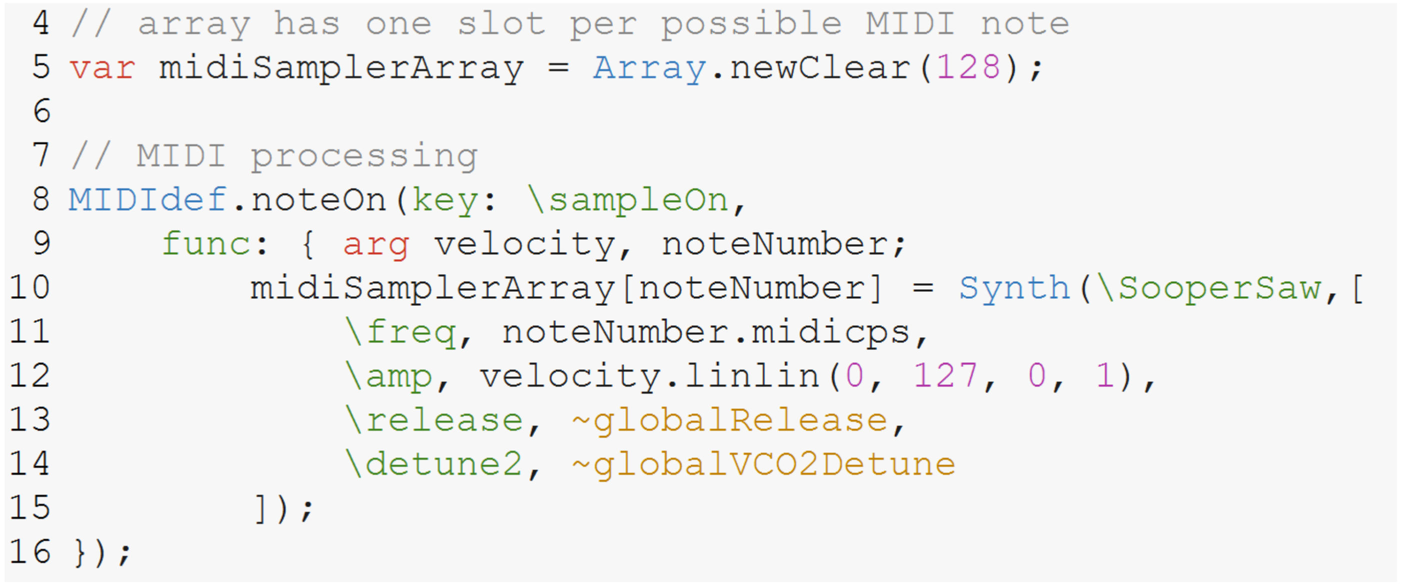
- A technical standard (Kakehashi, Smith, Wood et. al.)
  - communication protocol, digital interface, electrical connectors, file format
- Connect musical instruments and send (musical) data
  - note numbers, velocity, control data
  - 16 chanels
- Today → mostly MIDI via USB
  - still also possible with DIN cables (which does have some advantages :))

Connecting devices: MIDIIn.connectAll


episode_02_subtractive_synth_adding_midi.scd (~/Desktop/Computer-Music/supercollider_sketche) - SuperCollider IDE
File Session Edit View Language Server Help
episode_02_subtractive_synth_adding_midi.scd

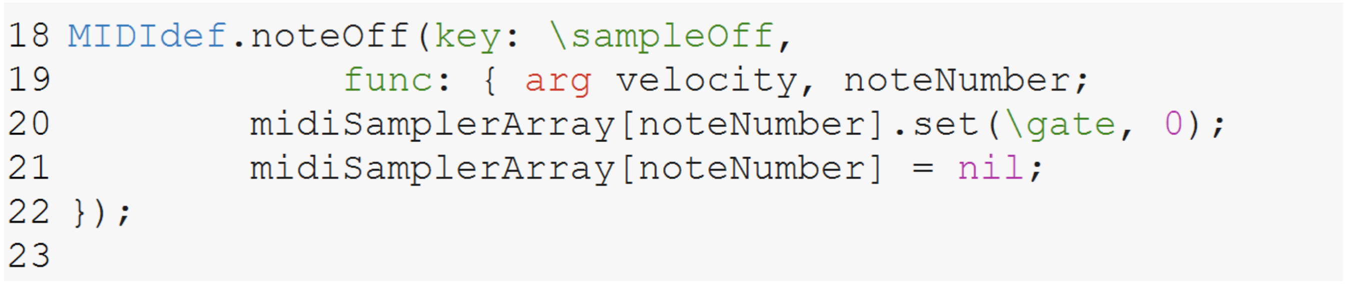
```
// midi part
MIDIIn.connectAll;
```

Callbacks: MIDIDef.noteOn



```
// array has one slot per possible MIDI note
var midiSamplerArray = Array.newClear(128);

// MIDI processing
MIDIdef.noteOn(key: \sampleOn,
    func: { arg velocity, noteNumber;
        midiSamplerArray[noteNumber] = Synth(\SooperSaw,[
            \freq, noteNumber.midicps,
            \amp, velocity.linlin(0, 127, 0, 1),
            \release, ~globalRelease,
            \detune2, ~globalVCO2Detune
        ]);
});
```

Callsbacks: MIDIDef.noteOff



```
MIDIdef.noteOff(key: \sampleOff,
            func: { arg velocity, noteNumber;
        midiSamplerArray[noteNumber].set(\gate, 0);
        midiSamplerArray[noteNumber] = nil;
});

```

Callbacks: MIDIDef.cc

```
MIDIdef.cc(key: \ccTest,
    func: { arg value, ccNum, chan;
        chan.post;" ".post;
        value.post;" ".post;
        value.linlin(0,127, 0, 5 ).post;" ".post;
        ccNum.postln;

        if(ccNum==74,{ ~globalRelease = value.linlin(0,127, 0, 5 );"gR".postln;});
        if(ccNum==75,{ ~globalVCO2Detune = value.linexp(0,127, 0.00001, 0.5 );"dt2".postln;});

        // update all synths
        midiSamplerArray.do({arg synth;
            if( synth != nil , {
                synth.set(\release, ~globalRelease);
                synth.set(\detune2, ~globalVC02Detune)
            });
        });
    }
);
)
```

Computer.Music.And.I

Conclusions

- SuperCollider provides support for processing Midi-Data
  - MIDIIn
  - MIDIDef
- *Callback-Functions* are defined for processing the Midi-Data
- The *State* of a Synth is stored in variables
  - using *~name* for defining a variable
- To implement a *polyphonic* synthesizer we use an array
  - for each Midi note number a Synth is defined when the key is pressed

Final Thoughts

- The concepts we have seen in this tiny example are very common in many music programming frameworks
- In a synth a number of loops run in parallel
  - Midi-Loop to receive/send/process Midi-Data
  - Audio-Loop to generate the sound
  - Control-Loop to process data that controls aspects of the synth state (we will this later)