Working with pch2csd – Clavia NM G2 to Csound Converter

Gleb Rogozinsky[1], Eugeny Cherny[2] and Michael Chesnokov[3],

[1] The Bonch-Bruevich St.Petersburg University of Telecommunications, Russia
[2] Åbo Akademi University, Finland
[3] JSC SEC „Nuclear Physics Research“, Russia
gleb.rogozinsky@gmail.com

Abstract. The paper presents a detailed review on the *pch2csd* application, developed for conversion of popular Clavia Nord Modular G2 synthesizer patch format pch2 into a Csound-based metalanguage. The Nord Modular G2 was one of the most remarkable synthesizers of late 90s. A considerable number of different patches makes Nord Modular G2 to be a desirable target for software emulation. In this paper we describe the pch2csd work flow, including modeling approach, so the developer may use the paper as a starting point for further experiments. Each model of Nord Modular's unit is implemented as an User-defined Opcode. The paper gives an approach for modeling, including description of ancillary files needed for the correct work. First presented at the International Csound Conference 2015 in St. Petersburg, the pch2csd project continues to develop. Some directions for future developments and strategic plans are suggested. The example of transformation of Nord Modular G2 patch into the Csound code concludes the paper.

Keywords: Nord Modular G2, converter, metalanguage

1 Introduction

In this paper we give the report on the current state of the pch2csd project and also describe the aspects of using our application. The pch2csd was first introduced to the Csound community at the International Csound Conference 2015, St. Petersburg Russia. At that time, the authors presented the results of their first experiments on converting Clavia Nord Modular G2 (NM2) patches into Csound code and demonstrated simple Csound code, automatically created as an output of the converter [1].

The NM2 was a quite remarkable software-hardware modular synthesizer from the late 90s. Its core is built on 4 DSPs and it is programmed via GUI while being connected to the computer. Then programmed, it can run standalone [2].

The NM2 patch consists of two main parts called voice part (VA) and global effects (FX). The VA is a subject of polyphony, comparing to FX part. The

greater the number of voices played simultaneously, the lesser number of modules can be used before overrun. NM2 uses fixed sampling rate of 96 kHz for audio signals and 24 kHz for control rate signals. The overall number of modules is around 200, including numerous generators, filters, mixers, switches, MIDI units, logics, sequencers and some effects.

From the beginning the pch2csd code was written in C. At the present time we are working at translating it to Python. Rewriting the converter in Python made the code much simpler and easier to maintain, improved cross-platform support, as well as provided better distribution through PIP. Prior to our work, the NM2 patch format (pch2) had been already described by [3]. There were also several projects related to alternative graphical interfaces for NM2 [4], [5]. Their repositories also include some useful data.

2 The pch2csd Work Flow

After the program starts the user is prompted to enter the valid path to pch2 file. The program checks the consistency of the following components: Csound opcodes, mapping tables, and input-output tables. The program outputs check results as a table to the terminal window. The output csd file is then created in the program's working directory.

2.1 Modeling Approach

Each NM2 unit should have a corresponding Csound model, placed in the */pch2csd/resources/templates/modules* as a txt file with the module ID, i.e. *12.txt*. The list of module IDs can be found in *mod_type_name.json* file. The NM2 units are modeled as Csound UDOs. Csound compiler reads the code lines from top to the bottom, which is completely different approach to graphical system of patching. Fortunately, Csound has a *zak-patching* system. It provides a given number of *a-rate* and *k-rate* buses to intercommunicate between different instruments. Those two spaces are independent from each other, comparing to NM2 patching system, where all cables are sequentially numbered. Comparing to typical behavior of Csound opcodes, where each opcode typically accepts some input data as some parameters and outputs the result(s), our zak-based UDOs do not have any outputs. After parameter fields our UDOs have an IO field, in which the numbers of buses in zak-space are listed.

Thus in our system we typically have

```
SomeOpcode aP1, kP2 [, ...], kIn1 [, ...], kOut1[, ...]
```

where *aP1* and *kP2* are some module parameters, *kIn1* is a number of k- or a-rate bus to read an input data from, and *kOut1* is a number of some k- or a-rate bus to send data to. Using described approach we are able to place the opcodes in csd file in arbitrary order, just like NM2 user can.

2.2 Values Mapping

Another important aspect is the values mapping. The NM2 modules have several different controller types of a number of ranges, i.e. audio frequency range, amplitude range, normalized values in the range from 0 to 1, delay time ranges, envelope stage durations, etc. The actual values of the controllers can be seen only when using the software editor. The NM2 patch file stores 7-bit MIDI CC values without any reference to the appropriate range. Thus we had to manually copy all data types, ranges and values. The actual values can be found in *value_ maps.json* file. We use specially commented mapping lines, which start with an @ symbol and list the tables to be used for each parameter of the module. I.e *112.txt* (*LevAdd* module) contains following lines:

```
;@ map s 2 LVLpos LVLlev
;@ map d BUT002
```

This means that the mapping table for the first parameter of *LevAdd* (the constant value to add to the input signal) is dependent on a second (*s 2*) parameter, which is the two-state button (table *BUT002*). The button switches *LevAdd* from unipolar to bipolar range of values. According to the choice, one of two possible mapping tables is selected (*LVLpos* or *LVLlev*). The order of NM2 parameters are fixed. Each line of a mapping file relates to the corresponding parameter of NM2 unit. The first symbol after keyword *map* marks whether the parameter needs direct (*d*) mapping, i.e. its 7-bit MIDI CC value points to the actual value in given table (i.e. *BUT002* table), or the actual parameter value depends on some selection (*s*). For the last option, there should be several possible mapping tables listed (*LVLpos LVLlev*) after the pointer to selector (*2*). Typically each module starts from one or several mapping lines. The mapping tables are stored in */pch2csd/resources/value_ maps.json*.

2.3 Polymorphism Issue

It should also be noted that several NM2 modules are polymorphous, i.e. k-rate filter can turn into a-rate filter depending on a type of an input signal. Unfortunately there is no direct indication of current module type in a pch2 file, so the actual module type can be found only through analyzing of incoming cable connections. Our algorithm checks the module type, and its input connections. In case of non-default type of the input, the corresponding module twin is used instead of the default one. As it mention above, the default module UDOs are placed in Modules directory. The twins are placed in *seludoM*[1] directory under the same name.

1 Reverse of the word 'Modules'

3 The Current State

We currently have 100^2 of 182 modules implemented as Csound user-defined opcodes and 36 mapping tables to map the values from the 7-bit ints to non-linear parameter range. Storing the data as the text allows anyone to change the behavior of the modules without touching the code.

Most of implemented modules still need careful checking and improvements. From the NM2 simulation side, most attention should be paid to generators. Almost every sound begins with the oscillator section, and the corresponding units should be modeled first. The NM2 oscillators produce aliased waveforms at 96 kHz. The Figure 1 shows the amplitude spectra of Clavia's sawtooth. We can clearly see the aliasing part of the spectra, mirrored from the Nyquist frequency (red line, 48 kHz). This feature of NM2 distinguishes it from the popular family of analog-modeling synthesizers, which typically produce alias-free waveforms, and makes it possible to simulate the corresponding waves by simple generation of ideal piece-wise functions using GEN7. The authors have already implemented most of the straight-forward modules, i.e. mixing operations, level, delay, logics and switching. Though some of them still need some careful approach for being as close as possible to their NM2 copies, i.e. *NoiseGate* and *EnvFollower*.

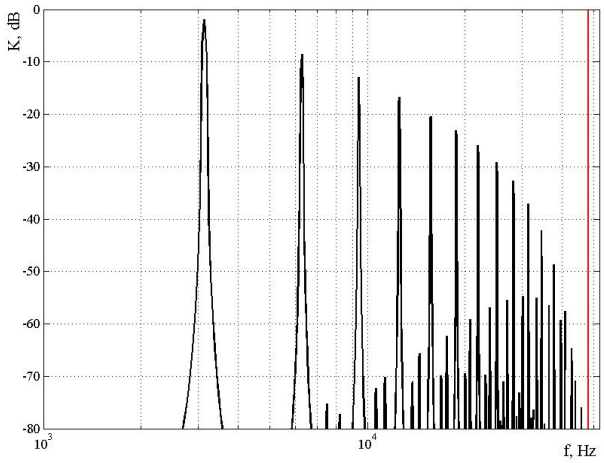

Fig. 1. Spectra of NM2 sawtooth wave.

The main problem regarding envelopes is the NM2's ability to change the envelope parameters during the run. This makes no use of Csound's typical solutions based on *linseg*, *expseg* or *transeg* units. Filters are rather the question of time, since their behavior should be carefully copied, i.e. through analysis of white noise filtering.

2 Of which up to 20 modules so far have almost exact behavior as original NM2

Another important goal we started working on recently is to make the project hackable, so users would be able to easily modify module implementations to contribute to the project or to modify sound for their own tastes. Our next possible to-do after providing a completely working solution is an integration with some existing Clavia patch editor. It would actually establish the new Clavia-based software modular system running on a Csound core. Also, the native Csound developments, i.e. Cabbage Studio, seem very promising in the context of further integration. Current pch2csd sources can be found on the GitHub [6].

Below we give an example of code for the NM2 patch on Fig.2. The patch consists of three slightly detuned sawtooth oscillators, mixer, envelope unit, LP filter and a simple delay in FX section.

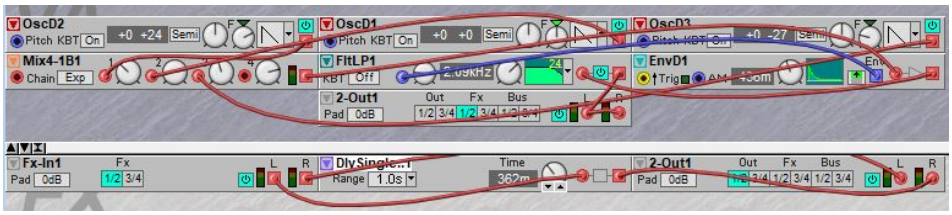

Fig. 2. NM2 patch used for testing.

Orchestra Example[3]:

```
sr = 96000
kr = 24000
nchnls = 2
0dbfs = 1

zakinit 12, 4  ; establish z-space of 12 a- and 4 k-rate buses

opcode OscD, 0, iKi  ; a simplified NM2 oscillator OscD model
   iPitch, kFine, iOut xin
   kfine = cent(kFine)
   icps = cpsmidinn(iPitch)
   aout oscili 1, icps*kfine, 1
   zaw aout, iOut
endop

opcode Mix41B, 0, kkkkiiiii  ; a NM2 4-1 mixer model
kLev1, kLev2, kLev3, kLev4, in1, in2, in3, in4, iout xin
   a1 zar in1
   a2 zar in2
   a3 zar in3
   a4 zar in4
   aout = a1*kLev1 + a2*kLev2 + a3*kLev3 + a4*kLev4
   zaw aout, iout
endop
```

3 Some ancillary code lines are intentionally omitted for the reasons of size

```
opcode EnvDSimple, 0, kiii ; a simplified NM2 envelope generator
   kH, iIn,iEnv,iOut xin
   aIn zar iIn
   kEnv expseg .00001,.001,1,i(kH),.00001
   zkw kEnv, iEnv
   zaw aIn*kEnv, iOut
endop

opcode FltLP, 0, ikkiii ; a simplified NM2 LP filter model
   iOrder, kMod, kCF, in1, imod, iout xin
   ain zar in1
   kmod zkr imod
   aout tonex ain, kCF+kmod*kMod, iOrder
   zaw aout, iout
endop

opcode Out2, 0, ii ; a simplified NM2 Out2 model
   iL, iR xin
   aL zar iL
   aR zar iR
   outs aL, aR
endop

opcode DlySingleA, 0, kii ; a NM2 delay unit model
   kTime, iIn, iOut xin
   ain zar iIn
   abuf delayr 1
   aout deltap kTime
   delayw ain
   zaw aout, iOut
endop

instr 1 ; this is a VA part of NM2 patch
   OscD 64, 24, 2
   OscD 64, 0, 3
   OscD 64, -27, 4
   Mix41B 0.336, 0.781, 0.430, 0.781, 2, 3, 4, 0, 5
   EnvDSimple 0.456, 5, 3, 6
   FltLP 4, 2050,2090,6,3,9
endin
instr 2 ; this is a FX part of NM2 patch
   DlySingleA 0.362, 9, 10
   Out2 9,10
endin
```

Score Example:

```
f1 0 16384 7 0 8192 1 0 -1 8192 0
f2 0 16384 10 1
i1 0 [60*60*24*7]
i2 0 [60*60*24*7]
```

References

1. Rogozinsky G., Cherny E., and Osipenko I.: Making mainstream synthesizers with csound. In: Proceedings of the 3rd International Csound Conference, pp. 132–140. St.Petersburg (2016)
2. Sound On Sound article on Clavia Nord Modular G2, http://soundonsound.com/reviews/clavia-nord-modular-g2
3. Michael Dewberry Home Page, http://www.dewb.org/g2/pch2format.html
4. NMG2 Open source editor, https://sourceforge.net/projects/nmg2editor/
5. G2Dev page, http://bverhue.nl/g2dev/?page_id=17
6. pch2csd GitHub page, https://github.com/gleb812/pch2csd