# Electric grammars. Algorithmic design and construction of experimental music circuits

**Simone Pappalardo**
Conservatorio Statale di Musica "O. Respighi" di Latina
simpapp@yahoo.it

**Andrea Valle**
CIRMA-StudiUm - Università di Torino
andrea.valle@unito.it

## ABSTRACT

The paper discusses a generative approach to the design of experimental electronic circuits for musical application. The model takes into account rewriting rules inspired by L-systems constrained by domain-specific features depending on electronic components, and generates families of circuits. An integrated production pipeline is introduced, that ranges from algorithmic design to simulation up to hardware printing.

## 1. INTRODUCTION

A generative approach has been intensely pursued in the digital domain, i.e. where a certain generative system has an established connection between software and final output. Computer music has worked extensively on algorithmic composition techniques that have been applied since its inception both to event organization and audio generation [1]. On the other side, physical computing [2] has successfully tried in recent years to fill the gap between computation and the physical world, by linking algorithmic design to a heterogeneous set of production systems, e.g. in the extensive applications of 3D printing. Concerning electronics, and in particular electronic circuit design, it can be observed that the latter is a complex task, as it involves accurate selection of components in relation to a specific desired output. Thus, algorithmic techniques are used on the design side mostly for circuit optimization. As an example, evolutionary algorithms have been used to design circuits: the rationale is that in some cases a certain performance is expected and the algorithm is iteratively run so to design a circuit to fit the targeted performance [3]. In this sense, generation is constrained by an expected result rather than focusing on the exploration of new behaviors (see also [4] for AI techniques based on an evolutionary computation for circuit design). Moreover, hardware layout for electronic components involves many critical steps, depending on both component features (packaging) and routing constraints (e.g. non overlapping of routes). Thus, a completely automated pipeline as required by a truly generative approach, linking algorithmic models to final physical output (i.e. electronic components) is a difficult task. On the other side, nowadays available technologies seem promising in favoring such an approach, as a panoply of Electronic Design Automation (EDA) software solutions are available, that can be directly linked to automatic circuit printing. In the following, we discuss a generative methodology for audio circuit design, propose an application, and describe an integrated pipeline solution from circuit modeling to PCB printing.



## 2. ALGORITHMIC DESIGN FOR AUDIO CIRCUIT

Generative approaches to art and creativity are typically intended as ways to yield complex structures by defining compact formal procedures. A plethora of models and techniques has been defined in the last decades, including, among the others, formal grammars, fractals, cellular automata [5]. As such approaches are formal, they have been applied/mapped to various aesthetic domains, including e.g. visual art and music composition. The rationale at their base is to shift the focus from the final work to the meta-level, that of a system capable of generating various "legal", "well-formed" outputs. In turn, outputs may be evaluated, e.g. filtered in terms of acceptance/refusal in relation to some (un)desired features, and the evaluation may be used to tune the system by means of a feedback loop. In short, algorithmic techniques like the aforementioned ones are mostly used to explore the output space of a system. In the domain of sound production, analog audio synthesis –far from being a dead end– has survived to digital audio and it is at the moment flourishing among musicians, in particular in hybrid systems that retain analog audio generation while assigning control to digital modules. In the field of electronic circuits for music applications, modular systems that can be assembled in various ways are a standard solution since the inception of analog electronic music, and recently various efforts have been made to create miniature, highly reconfigurable modular systems (e.g. littleBits [1] ). However, as the assembly is left to the user, all these systems are not suited for generative techniques. In this sense, the generative process should move to a lower hardware level. A generative approach to audio circuit design has not been investigated, as far as we know, as typical audio circuits implement well-defined schemes that are a priori constrained in relation to specific needs, e.g. noise reduction optimization or adherence to a certain already es-

[1] http://littlebits.cc

tablished model, as in the case of guitar pedal effects. On the other side, many experimental music practices focus on the manipulation of analog audio circuits with a clear exploratory aim [6]. As an extreme example, the popular practice of circuit bending [7] even proposes a paradoxical "anti-theory" as a knowledge background for a totally empirical approach to electronic hacking. Differently from the previous empirical approaches but sharing the same exploratory perspective, we propose a generative approach to audio circuit design for experimental music. The main aim of our project is to produce families of circuits, sharing some basic features, that can be printed automatically, so that each board is unique while retaining a family resemblance with the other from the same generative model.
The design methodology is a three-step process:

1. **Model retrieval**: as a first step, an already established audio circuit is taken into account, so to start with a well-know and working example;

2. **Functional decomposition**: the circuit is theoretically decomposed into functional units that can be thought as closed elements provided with inputs and outputs. These units can include various electronic component configurations. Granularity at the single component level (e.g. diode) is too basic to provide an operative base for generation, as too many constraints would have to be taken into account.

3. **Rule-based re-organisation**: functional units are considered as symbols to be arranged in strings following a specific formalization.

While conceptually separated, the three steps are tightly integrated, as there is a constant feedback loop among searching for suitable audio circuits, investigating about their fitting into a modular approach, and assessing the robustness in scaling up component assembly via generative procedure. In the following, we will discuss the previous methodology in relation to a case study.

## 3. MODEL RETRIEVAL AND FUNCTIONAL DECOMPOSITION IN A TEST CASE

In this section we discuss audio analog electronics, that is, in reference to the previous process, both models (1) and decomposition (2). We will discuss a specific test case to show various emerging issues, but the former is meant as an example of the previously introduced, general methodology. As already said, while it might be desirable to uncouple electronic circuit design from generative modeling, there is a feedback loop between the domain to be generatively modeled (electronic circuits) and the modeling system (see later). In general terms, our basic design choice was targeted to define an expansion procedure of a certain basic circuit. Indeed, other circuit designs may be taken into account as candidates.

Our aim has been to look for an already established audio schematic (methodology, step 1) provided with two features. First, it should be expandable by iterating a limited set of basic units, as the goal of the main electronic circuit

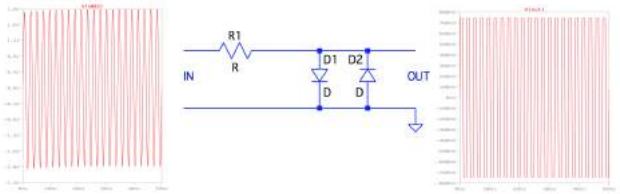


Figure 1. Diode clipping circuit.

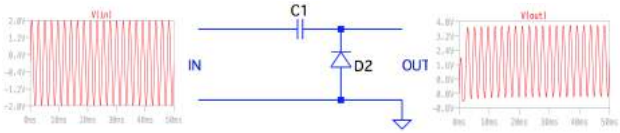


Figure 2. Diode clamping circuit.

is to link some independent object characterized by audio input and output (methodology, step 2). Second, it should provide a spectral enrichment (a music desideratum) without a final amplitude overload (an audio constraint related to generative procedure, as iteration results in stacking an undefined number of components).
In relation to step 1, it can be observed that diodes applications have been widely used in audio. in particular, diode clipping (Figure 1) is often used in analog and audio circuits to create distortion effects or to protect a device's input from possible current overloads: as a consequence, various schematics implementing this technique are available (e.g. [8]). Diodes used in analog effects for guitar generally clip when the input signal voltage reaches the forward threshold voltage of the component. For our circuit we decided to work with Schottky diodes so to have a very low clipping threshold (typically $0.2$ V against $0.7$ V of a silicon type).

In classical analog distortion guitar effects, the amplitude of the input signal is limited in either directions by a fixed amount. Indeed, in these devices, the reference threshold voltage for diodes is the circuit ground, so that the input signal clips at the small forward voltage of the forward biased diode. In practical applications, the threshold voltage of the diode, and thus therefore the signal clipping point, can be set to any desired value by increasing or decreasing the reference voltage [9]. It is therefore theoretically possible to modulate the clipping threshold of a wave by setting an audio-controlled variable threshold as the forward biasing of the diode. In our case, the threshold of the diode is controlled by a clamping circuit [10] (Figure 2). In order both to isolate the effects of two circuits (clamping and clipping) and to control impedances to implement a simulation of an audio-modulated diode with Spice3 [11], we improved the theoretical schematics in Figures 1 and 2 by introducing two operational amplifiers, to be used as voltage followers. The resulting circuit is shown schematically in Figure 3. It is still a theoretical circuit but sufficient to obtain a simulation of a "carrier" sine wave (220 Hz) dynamically clipped by another sinusoidal "modulator " (439 Hz). Figure 4 shows a new simulation of the

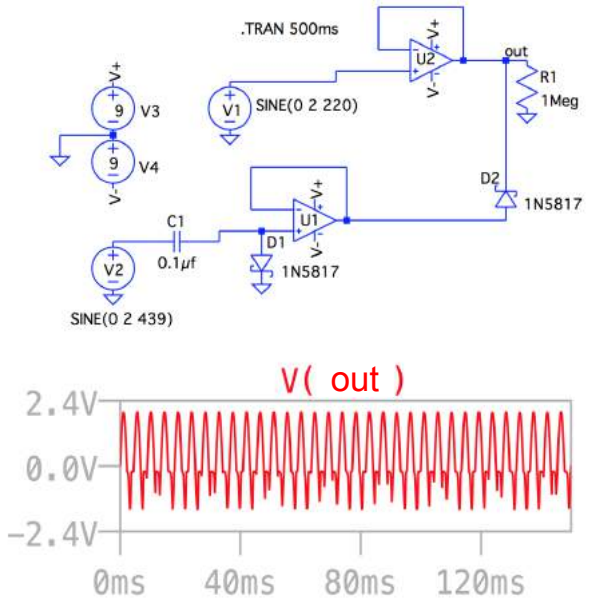


Figure 3. Half wave diode-modulated clipping (dmc)

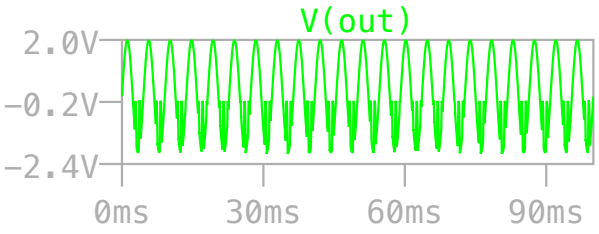


Figure 4. Half wave dmc with $C = 220$ Hz and $M = 1239$ Hz.

same schematic, this time featuring a 220 Hz carrier and 1239 Hz modulator. This schematic allows the modulation of the negative half-waves of a carrier sine by changing at audio rate the voltage reference at the anode of a diode. In order to modulate both negative and positive parts of the carrier sine wave, Figure 5 shows the implementation of a duplication/reversal of the clamping schematic. Such a clamping circuit is intended to limit a third modulating wave in the positive range. This signal is used to induce a fluctuation in the voltage at cathode of a further diode that modulates the positive part of the carrier half-wave.

Figure 5 shows a Spice3 simulation with positive modulator = 1123 Hz, negative modulator = 439 Hz, and carrier = 220 Hz.

The outlined schematic can be expanded by a systematic development, through a nesting and multiplication of modulating objects around the axis of the carrier signal, thus fulfilling the requirement of step 2 of our methodology, that prepares rule-base organization (step 3). In order to allow the nesting of elements, we took advantage of typical features of operational amplifiers. These components (op amps, as commonly referred), thanks to their impedance features and their ease of use, allow to simply cascade several parts, adding the possibility of frequency control by

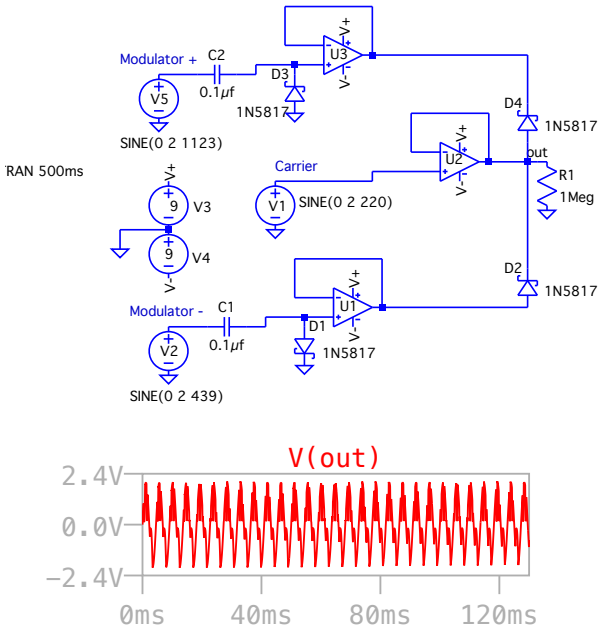


Figure 5. A modified version for full wave modulation dmc.

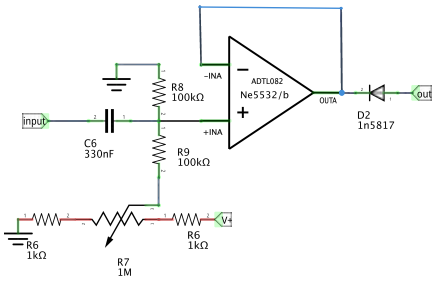


Figure 6. a) dc+.

means of inserting few capacitors. In our schematic model we add both a voltage follower object to use for coupling the various circuit each other, and an op amp input amplifier to filter, amplify and/or attenuate three input signals: carrier, negative modulator, positive modulator. The schematic model can be, then, decomposed into five units (see later): dc+, dc-, buffer, sig in, sig out.

Figures 6 to 10 show all the schematics of the final single objects. Figure 11 shows the complete root circuit ready for a Spice simulation. Colored blocks represent previously discussed components, that is, $a \rightarrow$ dc+; $b \rightarrow$ dc-; $c \rightarrow$ buffer; $d \rightarrow$sig in; $e \rightarrow$ sig out (for other letters, see Section 4, and the companion Figure 13). Figure 12 shows a physical test implementation of the "root" circuit, a minimal assemblage of the components that acts as starting state for generation.

## 4. RULE-BASED RE-ORGANIZATION

In general terms, many formalisms are available as generative models to govern an arrangement of audio components. In our case, we have explored L-systems [12]. L-

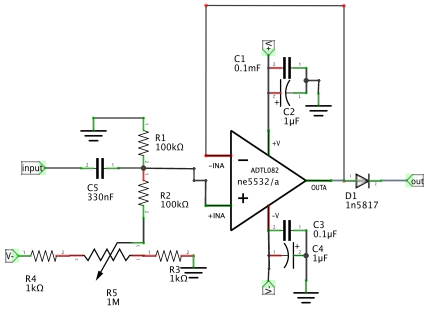


Figure 7. b) dc-.

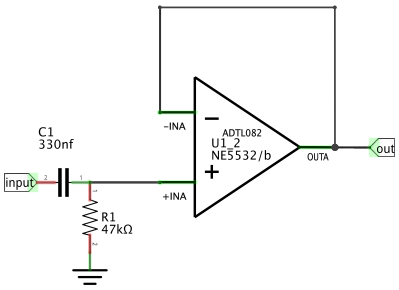


Figure 8. c) buffer.

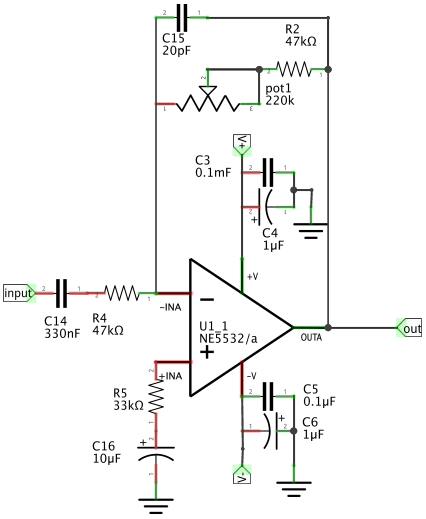


Figure 9. d) sig in.

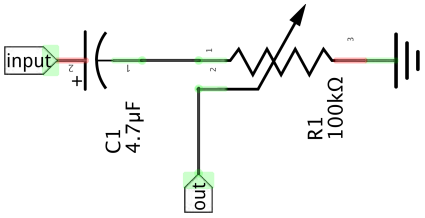


Figure 10. e) sig out.

systems have been proposed in biology as formal models to describe natural shapes that exhibit a clear generative behavior, i.e. an accumulation of features resulting from a growing pattern. They have originally proven successful in modeling plants, and then inspired algorithmic approaches to biological morphologies (shells, sponges, seaweed) [13] [14]. As formal systems, and apart from their interpretation in the biological domain, they have been widely used in generative art, including algorithmic music composition. In relation to our case, the choice of L-systems has been driven by considering an electric circuit, if not a tree, at least as a sort of vegetable rhizome. Hence the idea of recursively expanding a basic topology into a more complex network. Analogously to what happened in the model retrieval step, other formalisms may be explored.

L-systems are formal grammars in which a set of parallel rewriting rules ("production rules") is applied against an initial sequence of symbols (the "axiom"). Production rules allow to generate new sequences of symbols against which the former can be applied again. Even if the resulting morphology depends on the interpretation of symbols in a certain domain, the formal system structurally encodes features of a recursive organization. The following model for audio analog electronic circuit is inspired by L-systems. It can be defined as

$$E = (V, \omega, P)$$

where

$$V = \{a, b, c, d, e\}$$

is the set of symbols,

$$\omega = d*@1\ d*@2\ d*@3\ c1@4\ a2@4\ b3@4\ c4@5\ e5@*$$

is the axiom, and $P$ :

$$a \rightarrow c3@(k+1)\ b2@(k+1)\ c(k+1)@(k+2)\ a(k+2)@4$$
$$b \rightarrow c2@(k+3)\ a3@(k+3)\ c(k+3)@(k+4)\ b(k+4)@4$$

is the production ruleset.

As a variation on standard L-systems, the proposed model includes integers associated to each symbol in the form $i@o$, where @ is a syntactic separator. The symbol $k$ represents an integer associated to the complete rewriting $n$ (i.e. application of all rules), so that $k = n \times 4$. Empty spaces and brackets have only a typographical meaning for sake of readability. In the interpretation context, only the string resulting from the last rewriting is taken into account. Each symbol from $V$ is associated to the relative electric component (see Section 3). The symbols $i$ and $o$ are interpreted as labels respectively for input and output ports. Thus, a symbol like $b2@14$ indicates that the component $b$ has an input port labeled 2 and an output port labeled 14. The symbol $*$ represents a null value: it can be seen that components $d$ have a null input, as they are signal generators that feed the circuit, while $e$ has a null output, being itself the output of the system. In order to define the topology of the circuit, port labels are taken into account, and input ports are connected to output ones with the same label, the signal flowing from output to input. Graphical interpretation of the strings allows to inspect visually the resulting electronic topologies. The following graphs have been obtained by automatically scripting the dot language for graph visual-

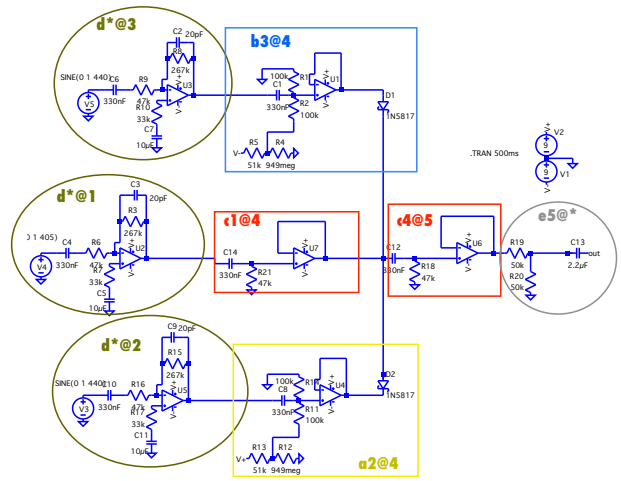


Figure 11. Root circuit (axiom).

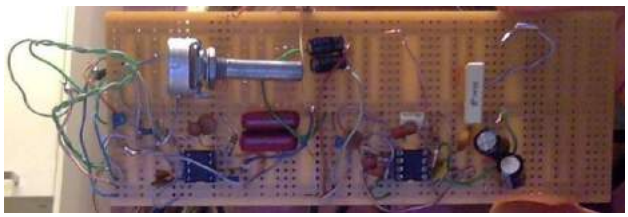

Figure 12. Root circuit, first test.

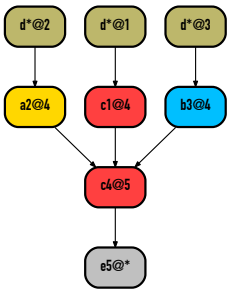


Figure 13. Axiom, with connection labels.

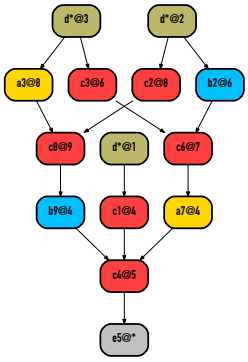


Figure 14. First rewriting, with connection labels.

ization [15]. Figure 13 shows the topology of the circuit of Figure 11 in relation to its modeled decomposition into the axiom $\omega$. Each component is colored in relation to its type, and labeled according to the mapped symbol, like in Figure 11. It can be seen that output ports are connected to input ports with the same identifier. Figure 14 shows the first application of the ruleset ($n = 1$).

The definition of the formal system has been driven by semantic constraints, i.e. by taking into account viable electronic connections. While increasing $n$, the resulting string shows some spurious artifacts, in terms of its electronic interpretation. Figure 15 shows the topological graph with $n = 2$ (labels have been replaced with the component types). It is apparent from Figure 15 that buffers at the graph' side are misplaced in the electronic interpretation, as they are isolated. To solve this issue, a recursive pruning strategy is applied to the graph, so that no isolated buffers are present in the final graph. Figure 16 shows the graph of Figure 15 after pruning.

## 5. AN INTEGRATED PIPELINE

While algorithmic modeling of electronic circuit topologies can have a theoretical interest, it is indeed very far from a practical application. The aim of a generative approach is indeed to obtain circuits whose complexity cannot be reached by hand design and that are (o should be) still guaranteed to work by the system definition itself. This requires an integrated automatic pipeline, ideally from formal model seamlessly to PCB printing. The implemented solution is shown in Figure 18. Grey boxes represent used softwares, while white boxes are internal blocks developed to perform specific functions. Plain text labels represent textual data. The generative model has been implemented in SuperCollider [16], an audio-targeted programming language that features high-level data collection manipulation. The SuperCollider component performs two task: generation and mapping. L-Generator is the module implementing the L-system generation: it takes an L-system specification as a textual input, and output a string (as a result of rewriting). The string is fed into the Parser that converts it into a data structure apt to be further manipulated by the Pruner module. The resulting data structure can then be mapped into other formats. The mapping task can be seen as a software gluing process. The DotGenerator creates a dot file to be subsequently rendered by the dot program as a visual file. As discussed, this is instrumental in rapidly understanding the behavior of the generation process. LTGenerator creates input files (in the "asc" ASCII format) for the LTSpice IV software [17]. LTSpice IV includes simulation of the circuit via Spice3 (including audio generation, a crucial features in our case) and schematic drawing. In order to generate asc files, component descriptions are stored in specification templates (spec file), in which opportune textual replacements for identifiers and positions are performed. Component circuits ($a$–$e$) are algorithmically placed on the board by following a simple carriage return strategy over a fixed width grid and using textual labeling of input and output as internal ref-

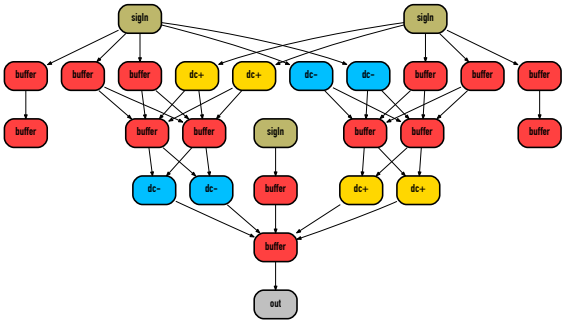


Figure 15. Second rewriting.

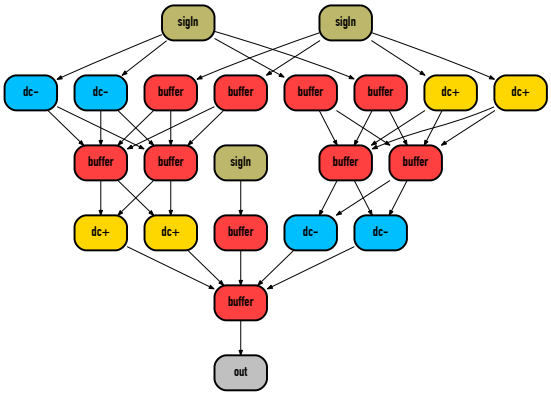


Figure 16. Second rewriting, pruned.

erences. Figure 23 shows the circuit of Figure 17, as converted by LTGenerator into asc format and drawn by LTSpice VI, with a zoomed component for sake of readability. Figures 20 and 19 show sonograms and waveforms from Spice 3 audio simulation of a very large circuit, obtained by four rewritings. It can be seen that complex spectra are easily obtained, with many partials in various harmonic relations. Spectra are sensitive to small parameter variations, thus ensuring a varied palette that can be exported as a user control (e.g. in the case of the three modulating frequencies, the latter can be associated to rotary controls on the final board). While Figure 19 illustrates a fixed spectrum, Figure 20 shows an evolving one, resulting from real-time parameter tweaking. While these results from extensive tests –matching the typical flavor of analog audio modulation but in a very rich way– are very encouraging, the main (in fact, surprising) drawback of simulation is that it has proved to be far from accurate (see later), so a real test phase has to be performed on the final hardware circuits. The asc format is also used as an interchange format to communicate with EasyEDA [2], a web-based EDA tool suite. EasyEDA has functionalities to facilitate (even if not to fully automate) routing for printed circuit board layouts and to directly manufacture printed circuit boards. In this way, it is possible to link the formally generated specification file to PCB printing. Figure 22 shows the root circuit PCB (i.e. the axiom) obtained by running the autorouting procedure in EasyEDA.

[2] https://easyeda.com/

## 6. OPEN ISSUES

At the moment of writing, we have tested physical circuits on breadboard, generated layout diagrams, run extensive simulations, while we have still not printed PCBs. Thus, we cannot evaluate the very final product of the pipeline. In any case, two main issues have emerged.

**Simulation vs. Reality**. For each object, various tests were made in order to find the best configuration in terms of matching between Spice3 simulation results and real physical behavior (tested on breadboard implementations). This is a crucial point, as –in order to simulate with a good approximation the circuit even after several expansions by the generative system– it is essential that the Spice simulation is very close to the actual behavior of the circuit. Otherwise, unpredictable, non-functional results may emerge in the printed circuit. As a matter of facts, real circuits assembled on breadboard have always shown relevant differences in their behavior if compared to Spice3 simulations. Three main factors account for these differences:

– the power supply quality which must be opportunely and recursively filtered, unlike in the simulated schematic;

– the extensive need, between input and output of several op amp, for decoupling capacitors, not necessary (at least in the same amount) in the simulated circuit;

– the placement of the components on the board, that may create ground loops, again absent from Spice.

The physical circuit is usually more sensitive to small, sometimes timy, variations in the starting conditions. Here, nonlinearity of the components seems to be more visible. For example, in our case, small variations in the signal amplitudes (modulating and/or carrier) result in large differences in the resulting output. The simulation in Spice seems to smooth out the nonlinearity of the components. In this sense, the diode clamping (circuits $a$-dc+ and $b$-dc-) is the most sensitive to small variations of the input signals. We tried physically different circuits to limit and optimize the results. in order to allow a more flexible modulation of the threshold of the diode we chose to use a circuit that allows us to select by means of a potentiometer the amount of dc-offset output. A different amounts of offset corresponds to a greater or lesser depth of action of the modulating diode.

**Pipelining**. Figure 21 shows an ideal, 5-step model. First, an abstract topology of the circuit, such as the ones in Figure 14-16, is generated (1). Then, the topology is mapped onto an actual electronic topology, incorporating data from real electronic components, but without any metrics referred to the final circuit layout, so that a simulation can be rendered (2). The electronic circuit should be rendered as a schematic in order to provide a standard visual feedback on the result (3). Finally, the circuit schematic should be converted into a description of the physical layout, including component packaging and routes (4), so to be delivered for PCB printing (5). Candidates for step 1 include substantially all available programming languages, better if shifted towards high-level due to data manipulation in the formal

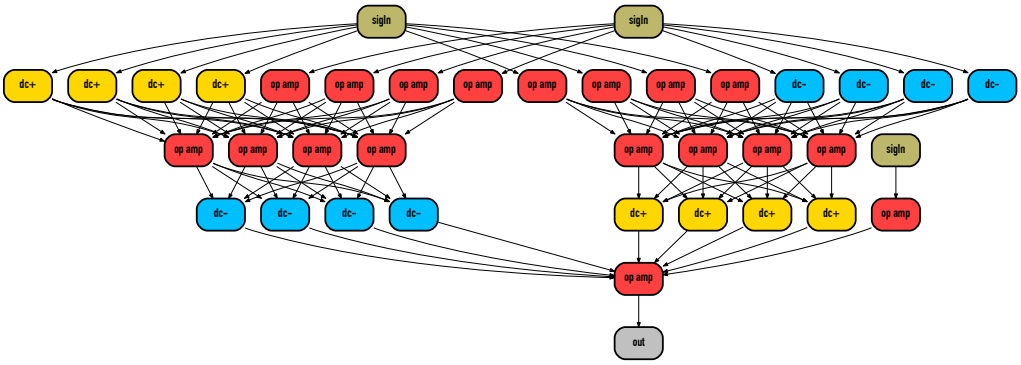


Figure 17. Third rewriting, pruned.

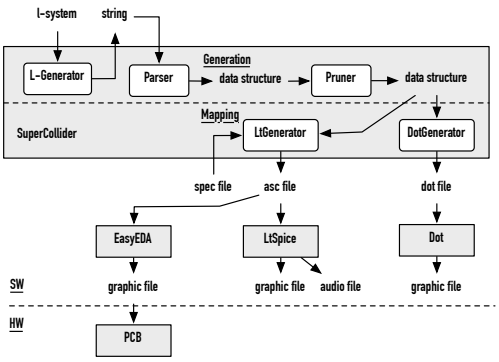


Figure 18. Implemented integrated pipeline.

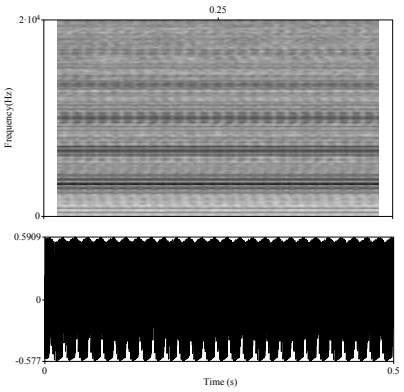


Figure 19. LTSpice audio simulation with 4 rewritings, $55, 44, 3345.7$ Hz.

model. A standard candidate for step 2 is indeed Spice3, that takes as its input a so-called netlist, i.e. a textual specification of components and connections, and is able to run simulation (including audio) and perform various analysis. Various software packages are available for steps 4 and 5, like e.g Autodesk Eagle. Actual available softwares for electronic layout are intended as computer-aided software for interactive editing toward the final layout organization, not as automated tools. A critical issue in relation to the ideal pipeline is that all the data should automatically transit from one step to the other. As an example, our choice (EasyEDA) has been determined by file format compatibility with LTSpice IV. On one side, schematic capture soft-

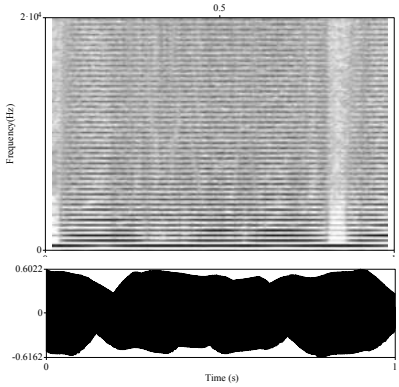


Figure 20. LTSpice audio simulation with 4 rewritings, $439, 440, 442.5$ Hz.

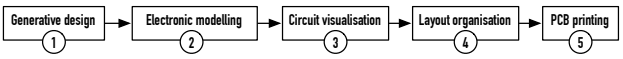


Figure 21. Ideal pipeline.

wares (as required by 4) are intended for design, and includes Spice3 as a simulation engine, so that typically they are able to export netlist files but not to import them. This depends on the fact that such a passage would require automatic layout for drawing circuit schematic. In our case, we solved by defining a grid algorithm. On a different but correlated side, EDA softwares (5) aim at computer-aided layout design, but not at a fully automatic one. Autorouting facilities are available, but, as final physical layout depends on such a large number of parameters, automation is only partially possibles. Of course, very large, automatically generated circuits still require a large amount of hand-operated fine tuning of the layout before printing.

## 7. CONCLUSIONS

While still at a prototypical stage, we believe that our approach has shown potential interest for experimental computer-design of audio analog synthesis, as can be seen from resulting signals in Figure 20 and 19. Generative models allow to describe not circuits, but families of circuits, that share some features but may differ one from each other by details. Thus, with automatic PCB routing, it could be pos-

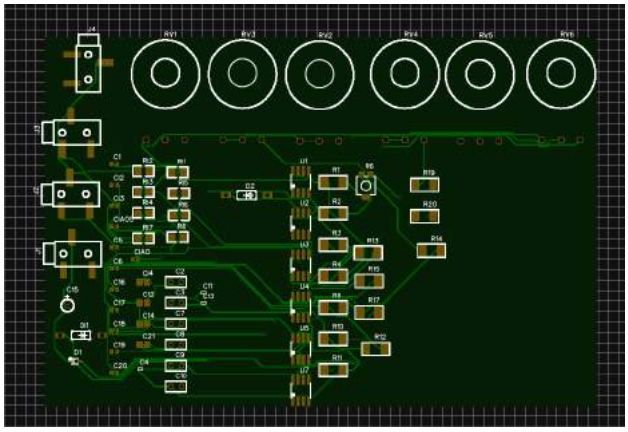

Figure 22. PCB for root circuit via autorouting.

sible to create singleton, unique synthesizers/processors. Indeed, the most delicate point is the layout for PCB printing of large generated circuits, that is still not robust from an automation perspective. But, once fine-tuned in relation to real physical behavior, circuits can be simulated, and only the final step (5) requires handcrafted operations. As a future work, we are planning to print PCB boards and test real final complex circuits, to analyze other analog circuits in order to decompose them, so that other generative models may be taken into account.

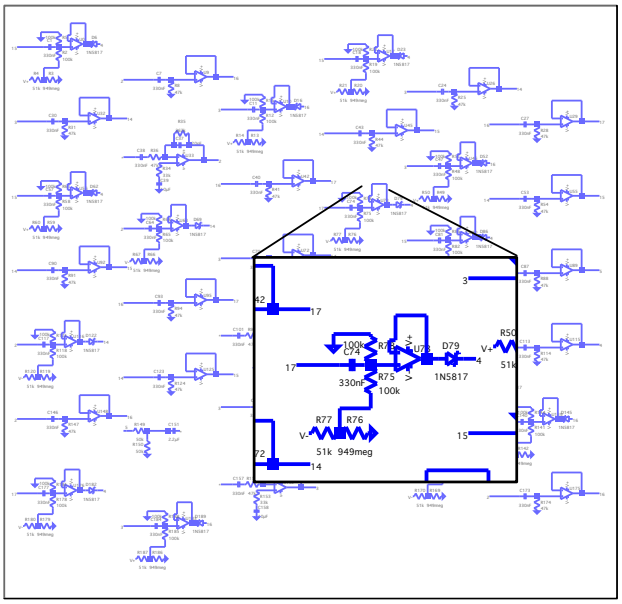


Figure 23. LTSpice drawing from automatic generated asc file, third rewriting, with a zoomed module.

## 8. REFERENCES


[1] C. Roads, *The Computer Music Tutorial*. Cambridge, MA, USA: MIT Press, 1996.

[2] D. O'Sullivan and T. Igoe, *Physical Computing. Sensing and Controlling the Physical World with Computers*. Boston, Mass.: Course Technology, 2004.

[3] M. M. B. Vellasco, R. S. Zebulum, and M. A. Pacheco, *Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms*. Boca Raton, FL, USA: CRC Press, Inc., 2001.

[4] A. Thompson, P. Layzell, and R. S. Zebulum, "Explorations in design space: Unconventional electronics design through artificial evolution," *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, vol. 3, no. 3, pp. 167–196, 1999.

[5] E. R. Miranda, *Composing music with Computers*. Burlington, MA: Focal Press, 2001.

[6] N. Collins, *Handmade Electronic Music. The art of hardware hacking*. New York–London: Routledge, 2006.

[7] R. Ghazala, *Circuit-Bending. Build your own alien instruments*. Indianapolis: Wiley, 2005.

[8] R. Salminen. (2000) Design Your Own Distortion. [Online]. Available: http://www.generalguitargadgets.com/how-to-build-it/technical-help/articles/design-distortion/

[9] P. Horowitz and W. Hill, *The Art of Electronics*, 3rd ed. New York, NY, USA: Cambridge University Press, 1989.

[10] VV.AA. (2012) DIY Circuit Design: Waveform Clamping. [Online]. Available: https://www.engineersgarage.com/tutorials/diy-circuit-design-waveform-clamping?page=3

[11] T. Q. A.R.Newton, D.O.Pederson, and A.Sangiovanni-Vincentelli, "SPICE3 Version 3f3 User's Manual," Department of Electrical Engineering and Computer Sciences, Berkeley, CA, Usser's manual, 1993.

[12] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*. New York: Springer, 1990.

[13] J. A. Kaandorp and J. Kübler, *The Algorithmic Beauty of Seaweeds, Sponges, and Corals*. New York, NY, USA: Springer-Verlag New York, Inc., 2001.

[14] H. Meinhardt, *The Algorithmic Beauty of Sea Shells*, 3rd ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.

[15] E. Gansner, E. Koutsofios, and S. North, *Drawing graphs with dot*, 2006.

[16] S. Wilson, D. Cottle, and N. Collins, Eds., *The SuperCollider Book*. Cambridge, Mass.: The MIT Press, 2011.

[17] G. Brocard, *The LTSpice IV Simulator: Manual, Methods and Applications*. Künzelsau: Swiridoff Verlag, 2011.