Abstract practices

Andrea Vogrig
Bachelor Thesis

Institute of Sonology
May 25th, 2018

Abstract

The thesis concern an open concept of abstraction in sound synthesis processes. Some inspiring works characterised by peculiar digital approaches to sound will be presented. The second chapter will deal with the development and fabrication of an open-source module called ‘Duad’, whose aim is to extend ways to control sound synthesis parameters. Various ideas and strategies have been implemented during Duad’s development, some of them will be discussed in details along the thesis. The last topic is my own artistic work, which explores some of Duad’s possibilities in fixed-media compositions and its practical integration into a modular synthesizer system.

Acknowledgements

I would like to first thank my family and friends for supporting and encouraging me to continue with my studies. I would also like to thank my mentor Lex van den Broek that patiently helped me with my electronic difficulties. Thanks to Bjarni Gunnarsson for helping me in the process of writing this thesis. Thanks to Peter Pabon, Johan Van Kreij, Raviv Ganchrow, and Kees Tazelaar for helpful and inspiring lessons. I want to thank all my colleagues at EWP in particular Catherine Ostraya, Giammarco Gaudenzi, Mari Mako, Marko Uzunovski, and Paul Schenkels for a lovely working experience. I would particularly like to thank Aisha Pagnes, Marcello Ghilardi, Ruben Brovida, Sarah Atzori, Tiziano Teodori for inspiring and supporting me along my studies.

Table of contents

Introduction

In today's musical scenario hybrid instruments are expanding our creative dashboard, giving composers access to new compositional tools and practices. The urge to gain freedom in the abstraction of compositional processes led me to research ways to control sound. Modular synthesizers provide a fertile ground for an empirical exploration where the modelling of abstractions can be exploited.

During the process of creating a new musical instrument the builder is allowed to choose, consider, tune or not tune all the details and technical aspects and possibly appreciate its results.

This research presents an instrument in the form of an Hybrid Eurorack module, an open-source programmable control voltage generator (processor) that can help in the exploration of various control voltage possibilities. The name of the module is *Duad*. This module is just one of the possible incarnations of a more generic control voltage breakout[1] board. The context of this investigation focuses on a hybrid interaction between a discrete digital procedure and continuous sound processes. The research will exemplify how *Duad* expresses complex behavior through standard and non-standard approach to synthesis. *Duad* implements numerous programs that allow to cover a wide range of modulation possibilities. Some of the functionalities that drive *Duad* will be analysed in detail, namely Bo and Op.

Bo is a particle system used to generate complex dynamic behaviors through collision detection where simulated physical laws can be distorted at will (see chapter 2.6). Op explores a 'non-standard' approach to synthesis (see chapter 2.7) where mathematical operations and binary manipulations are inspired by Supercollider operators, Bytebeat formulas (see chapter 1.3), and esoteric programming languages. These types of operations produce different control structures that have been mainly explored in the context of a small modular synthesizer. This synthesizer consists of a single analog variable state filter, used as main audio generator, on the top of that two low-frequency oscillators (LFO) and *Duad* are combined to explore various control voltage possibilities. In the following pages, I aim to investigate how certain digital approaches to sound could lead to interesting compositional practices. This investigation will be supported by an overview of the ideas that triggered the development of *Duad*. The resulting strategies for integrating *Duad's* functionality into actual composition will be elucidated.

[1] Denoting a group or unit/module that breaks away from a larger system.

Chapter 1

Digital reactions

Various experiments with hardware and software-setups have been explored during the last four years. The common ground between these experiments consists of various ideas which focus on the control of sound processes. Independent computer programs and small devices have been designed to investigate alternatives to a traditional control of sound parameters and mapping strategies. These programs have been firstly developed and tested in confined situations where OSC (Open Sound Control) messages have been used to control and interact with continuous sound procedures (see Appendix A) .This exploration moved into the context of an electronic modular synthesizer where hardware and software are combined in a self contained modular unit that embraces diverse control signals approaches.

1.1 Signals

A signal, in its physical form, is generated from a specific point or source and propagates through a transmissive medium that affects its physical property. "*Signal as referred to in communication systems, signal processing and electrical engineering is a function that conveys information about the behavior or attributes of some phenomenon."* (Priemer, 1991, p. 1). Signals play a fundamental role in any form of communication present in our world. We are surrounded by signals of every kind: signals have a different name and dimension depending on the context in which they are considered. Sound is a signal perceived by one of our major senses. Music along with all other forms of sound, consists of mechanical waves traveling through air.

"*A musical instrument is an instrument created or adapted to make musical sounds. In principle any object that produces sounds can be considered a musical instrument.*" (n.d.).

Electronic music instruments or synthesizers are made up of a number of circuits, which can create or influence a signal in various ways. "*No sound exists within the synthesizer…only electrical signals flow through the circuits, when eventually these signals are fed into a loudspeaker they produce sound.*" (Horn, 1994 p. 1). There are two main types of signals used within a modular synthesizer: **audio** signals and **control** signals. Audio signals are 'audible' and can be heard by the musician. The frequency range of an audio signal varies between 20Hz and 20KHz. Control signals are used to drive or control audio signals and consist in general of much lower frequency. Control signals can be so low that they can not be heard. In digital circuits various subcategories of control signals can be derived: **logic** signals such as clock, gate and triggers.

Logic signals consist of square waves with only two possible states representing either 0 or 1 (HIGH or LOW). A digital circuit refers to a device that works in binary. A so called digital abstraction is adopted to represent 0 or 1 and is usually implemented by electronic devices that operate in a physical world in which there are no 0s nor 1s. Digital signals are finite numerical representations of continuous analog signals. At the basis of computer operations the application of binary Boolean algebra[2] gets implemented as logic gates by means of digital electronics. These sets of elementary logical interactions allow devices and modules to cooperate while executing complex tasks such as generating or controlling sound processes.

In software engineering and computer science, "*abstraction is a technique for hiding complexity of computer systems by preserving information that is relevant in a given context, and forgetting information that is irrelevant in that context.*" (Guttag, 2013). Data abstraction and its control involves the use of subroutines and control-flow procedures which allow to define data in meaningful ways. This research is motivated by such fundamental control-flow ideas and experiments regarding abstraction.

Digital representations can be of different types (datatypes), capacity / resolution. For programming purposes it is common to group sequences of bits into larger entities that take values in much wider ranges than those allowed by a single bit. Datatypes are useful abstractions used to define how data should be interpreted by a computer. Within the context of a modular synthesizer signals are expressed in voltage. A varying voltage is converted into a numerical representation within discrete boundaries. Analog to digital converter circuits (ADC) are used to convert incoming analog signals within different resolutions (8,16, 32 bit); a digital to analog converter (DAC) performs the reverse function, it converts a digital signal into an analog signal. In *Duad*, the signal is represented by a 16 bit variable that contain a number in the range from 0 to 65535 (2^{16}-1) corresponding to -5 to 5 Volt. In this quantized yet explorable numerical representation interesting dynamic phenomena can be observed. Within this numerical definition one can apply any possible signal manipulation and compositional idea. Any digital abstraction, implemented as a numerical model can potentially be re-contextualized in a musical form.

1.2 Non-standard approach

With the introduction of computers and softwares, new computer music compositions were created. During the 1960s software families such as Music-N[3] used to model acoustical instrument represented as streams of signals, printed on a magnetic tape and used by composers in the studio. This software among others at that time, was adopting a standard approach to synthesis, similarly today most of the available music softwares still do.

"*Standard approaches are characterized by an implementation process where, given a description of the sound in terms of some acoustic model, machine instructions are ordered*

[2] **Boolean algebra:** *"is the branch of algebra in which the values of the variables are the truth values true and false."* (Boole G. 1847 *The Mathematical Analysis of Logic*)

[3] **MUSIC-N** *"refers to a family of computer programs for generating digital audio waveforms through direct synthesis."*

in such a way so as to simulate the sound described" (Holtzman, 1979)."Non-standard synthesis" (a term coined by Holtzman) refutes the simulation of real acoustic models by making use of abstract procedures to generate digital sound unique to the imperative state machine.

One of the first non-standard approaches to computer music was realized by Gottfried Michael Koenig in the Philips laboratories in Utrecht in 1972 with the development of SSP (Sound Synthesis Program) a real-time system that uses amplitude and time values as the only raw materials (Koenig, 1971). During an interview in 1978, Koenig said about SSP: "*My intention was to go away from the classical instrumental definitions of sound in terms of loudness, pitch and duration and so on, because then you could refer to musical elements which are not necessarily the elements of the language of today. To explore a new field of sound possibilities I thought it best to close the classical descriptions of sound and open up an experimental field in which you would really have to start again.*" (Roads, 1978 p. 62).

Another notable figure was the programmer Paul Berg who (in 1976-1977) wrote the PILE compiler for the PDP-15 minicomputer at the institute of Sonology in Utrecht, "*to hear that, which without the computer could not be heard; to think that which without the computer would not be thought; to think that which without the computer would not be learned*" (Berg, 1979). PILE is a computer language for sound synthesis that abstracts low-level machine operations as instructions. PILE manipulates numerical systems to program distinct sounds and structures. A PILE program does not refer to a particular physical model nor acoustic parameters (i.e.: frequency, timber and amplitude). It can produce up to four simultaneous output signals and operates in real-time once compiled. (Berg, 1979 p. 30)

Various names have been used to describe these types of numerical manipulations in musical context. In 1996 Curtis Road suggested the term *'Instruction synthesis'*. More recently Luc Döbereiner refers to this approach as *"Compositionally Motivated Sound Synthesis"* (Döbereiner, 2009) whereby composition and sound synthesis afford one another of interdependent possibilities. What interests me regarding the aforementioned composers is their ability to think beyond conventional production schemes by introducing new and practical ideas for the process and organization of sound events, resulting in a pioneering artistic exploration determined by the design and structure of the technology implemented. Non-standard approaches to synthesis shifted the traditional compositional paradigm based on loudness, pitch and timber into an imaginative exploration of sound parameters. The use of simple numerical manipulations to control and produce sound structures intrigues me for various reasons. Non-standard approaches are characterized by simplified numerical models that can operate in real time, these models procedurally calculate the next output sample by means of elementary arithmetic and binary operations. These operations are used to produce complex waveforms or control structures that do not necessarily relate to known acoustical phenomena. The exploration of these methods is based on an empirical and direct approach to sound abstractions that allow to extend and discover new interesting sound possibilities. Non-standard forms of manipulations can be easily explored in the context of simple and compact computer programs and are applicable in various sound domains.

1.3 Bytebeat

'Bytebeat' is a type of very short computer program that generates music. In 2011 Ville-Matias Heikkilä (Viznut) was releasing a video on Youtube (Heikkilä, 2011a), presenting seven short C-language programs and their musical outputs. The video gathered interest inspiring many programmers to experiment on their own and share their formulas. These programs received considerable attention because they seem to be too short for the complex musical structures they generated. (Heikkila)

Heikkilä states: *"A Bytebeat formula is a simple arrangement of digital-arithmetic operations that have been elementary to computers since the very beginning. It is apparently something that should have been discovered decades ago, but it wasn't"* (Heikkilä).

In 6 December 2011 Ville-Matias Heikkilä released a paper "*Discovering novel computer music techniques by exploring the space of short computer programs.*"(Heikkila, 2011b) in which he discussed the programs resulting from tests made by dozens of individuals within various online communities. The paper highlights some rather unusual methods they use for synthesizing sound and generating musical structures. *"How and why Bytebeat programs work was often a mystery even to their discoverers.*" (Heikkila). Also when some theory about them was devised, it was often quite difficult to understand or apply these as the often esoteric use of arithmetic in Bytebeat surely doesn't aid its demystification. The Bytebeat formulas are analogue to the processes used in the early computer music experiments, where a similar rudimentary approach based on basic digital-arithmetic operations can be found in the waveform generators PILE and SSP developed by Berg and Koenig respectively. The analogies however do not follow up in aesthetic terms.

Non-standard types of numerical manipulations procedurally generate waveforms as results of calculations over time. These generators use only time and amplitude values to produce the output and does not necessarily produce correlated audio signals.

The Bytebeat formulas are based in an explorative process initiated by the following short C program:

```
int main() {
    for(int t=0;;t++)
        putchar(EXPRESSION);
    return 0;
}
```

The simplest possible waveform that we can generate with this type of program is a sawtooth. A simple for(;;) putchar(t++) generates a sawtooth wave with a cycle length of 256 bytes.

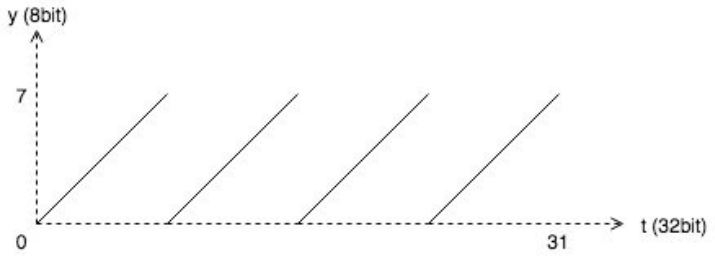


Figure 1: for(int t=0;;t++) putchar(t);

In the example shown (Figure 1) the expression is evaluated with 32 bit or more of integer precision while the *putchar*[4] function outputs only the eight lowest bits of each result. This binary truncation warps the *int* variable into a smaller variable while the evaluation procedure uses the full 32 bit field to generate a number. "*To control the frequency or pitch of the sawtooth oscillator the expression can be changed as follows: t++*2 is an octave higher, t++*3 goes up 7 semitones etc.*" (Heikkilä, 2011b). From this short C program different formulas have been discovered, each of which present their own particular behaviors and characters. More complex formulas use progressively more peculiar combinations of binary operators whereby through a process of trial and error ever more interesting formulas are discovered. ByteBeat is mostly used to produce a raw direct audio output, often resulting in pleasant musical structures with a 8 bit timbre. In this thesis I will describe how Op uses a similar idea to generate a control signal output (see chapter 2.7).

1.4 Modeling behaviors

The design of digital models that are used to control or generate sounds have been the interest of a wide community of scientists and composers throughout history. A model may help to explain or predict behaviors of a system like in natural science and to study the effects of its different variations. "*Modelling can often contribute to a deeper understanding of physical reality[...]. Moreover, the simulation can point out behaviours that have not been or even cannot be observed."* (Bellomo, Preziosi 1994). In physics it is common to idealize models for simplification. Within an abstract model responsible for the control of acoustic parameters, rules are not limited by strict realistic representations, instead a control model can consider abstractions which do not exist in representative simulations of reality. The idea of a control model, as opposed to producing sound directly, extends its possibilities by introducing virtual behaviors in order to control a synthesizer patch. When dealing with the design of an acoustical physical model, the sonic outcome tends to be observed from a purely scientific point of view, where the realism of physical simulation is sought.

[4] **putchar** "is a function in the C programming language that writes a single character to the standard output stream, stdout." It results in an 8 bit unsigned **char** variable in the range from 0 to 255. (*ISO/IEC 9899:1999 specification*. p. 299, § 7.19.7.9.) http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf

The design of an abstract model is purely based on virtual empirical experiments and organizational methods where the control of synthesis parameters can be considered in a wider compositional context allowing new sound discoveries. The models that I have developed are used to interact with a continuous sound process creating a more complex system from which sounds are generated. Ordered and chaotic structures emerge along the execution of these abstractions. Motivated by the curiosity of exploring rich dynamical interactions in a musical context, Yyyyttt explores these ideas by virtue of an empirical process and implement effective strategies that could potentially help me in the organization of sounds events. The need for such exploration comes from the desire of conceiving uncommon features that could extend the modulation possibility of a synthesizer. Using a mouse, keyboard, joystick or USB sensor board to control your modules and installations could be just a few of the many examples of possible uncommon applications (see chapter 2.4).

1.5 From model to sound

The research focuses on the design of hybrid abstract models as explorative practices into sound compositions. Its aim is to combine elementary signals operations used as basic transformation models for control signals (see chapter 2.7). The design of such an abstract digital model has been considered in the context of a small modular synthesizer, which consists mostly of voltage-controlled filters and low frequency oscillators. (see Appendix G) Although these types of transformations have been previously investigated within the context of a computer, the need to map a large set of parameters unavoidably characterized this investigation with physical instrumental limitations. An instrument needs to be small and portable and each controllable part should be physically accessible. Modular systems tend to grow fast. Various units or modules are generally required to control and produce sound, often extended cases and multiple power supplies are needed. An instrument should help and extend musicians' possibilities, allowing immersive explorations not limited by the setup configuration. Limiting the amount of modules effectively restricts the adjustable sound parameters available that can be simultaneously controlled and allows confined yet explorable parameters mappings. Experimenting with a limited set of modules and modulation possibilities often allow to produce raw frequency modulations that characterize the generated materials. This approach generally focuses on the micro-scale level of the composition, where diverse sound material is generated along an extensive exploration of the various connections afforded by the instrument's models. Confined patch configurations with slight variations are then observed and recorded. These models have so far been used to produce harmonically rich material with specific rhythmic patterns, serving as prime material for some of my compositions (see chapter 3.2). Although the models can also be used to organize sounds events in a macro-scale level to control the entirety of the composition, this generally emerges from the characteristics of the generated sound materials.

Chapter 2

Duad

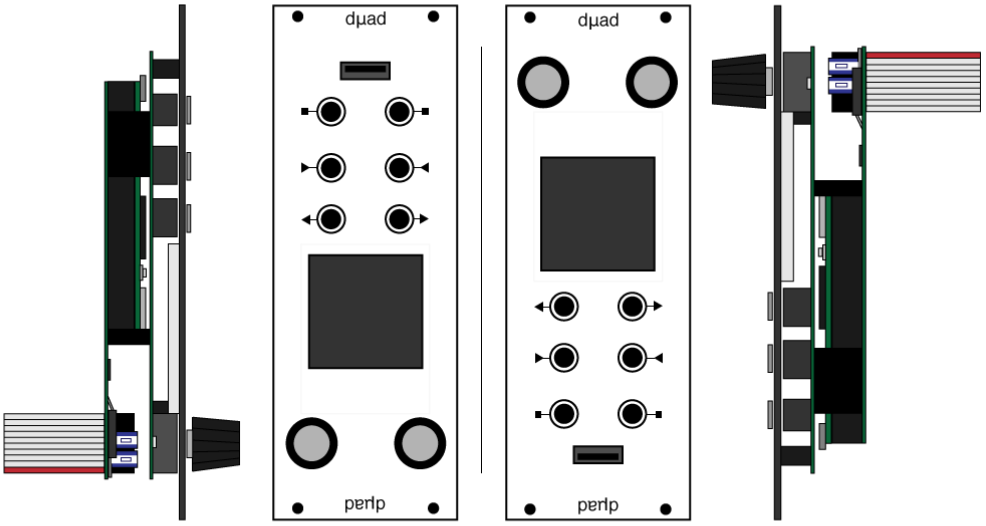


Figure 2: ***Duad***

Duad is a dual function control voltage generator. It has been developed during the third and fourth year of my studies at the Institute of Sonology in The Hague. Primarily the module serves as a bridge connecting digital thinking to the analog world. Secondly It opens almost endless forms of programmable interactions used to organize and process analog signals. One of the main goals is to explore how certain digital practices apply in an analog musical context. I have always imagined this module as being reprogrammable and extendable according to my needs, for instance allowing to select which algorithm is currently running within a set of available algorithms / programs. *Duad* implies the use of already existing platforms and devices. This chapter will mention some of the inspiring modules and techniques that have been considered during its development.

2.1 Microcontroller

An essential component of a digital module is the micro-controller which represents its programmable brain. In digital electronics the microcontroller[5] unit (MCU) is an electronic device designed to interact directly with the outside world through a program that resides in its internal memory by the use of specialized pins configurable by the programmer. There are three levels of processing capacity: 8, 16 and 32 bit. MCU's are generally used in embedded system and for special applications of digital controls. A notable example is the Arduino, an open-source prototyping platform, based on easy-to-use hardware and software.

[5] **Microcontroller** *"can be considered a self-contained system with a processor, memory and peripherals and can be used as an embedded system"* (Heath, 2003).

Arduino was born at the Ivrea Interaction Design Institute in 2005 as a low cost, easy tool for fast prototyping, aimed at students without a background in electronics and programming. The electrical engineer Paul Stoffregen raised founding for the development and fabrication of the Teensy development Boards family, a 32 bit ARM[6] microcontroller programmable in the Arduino IDE[7] (Integrated Development Environment). The Teensy controller family extends the possibility offered by 8 bit microcontrollers such as the Arduino. *Duad* design is based around the Teensy 3.6 microcontroller which offers a wide range of features. The module uses a 32 bit processor, multiple SPI[8] (Serial Peripheral Interface) buses and a USB host port to connect external MIDI/Serial devices.

2.2 Control Voltage

Control voltage (also known as CV) is an electrical signal used to manipulate analog circuits. Voltage control was a major step forward in the development of the synthesizer, prior to that time synth parameters were operated manually by switches and knobs. Voltage control can be used to control and affect parameters of other modules such as changing the frequency or amplitude of an oscillator, controlling the cutoff frequency and resonance of a filter and other infinite possibilities. *"The effect of control voltage depends entirely on how it is applied and not by some pre-established role [...] The user is free from the logic of suggested techniques and can abandon them in search of new, novel, surprising discoveries and functionalities."* (Hordijk, Rob). The concept of CV was fairly standard on analog synthesizers, but its implementation was not. *Volts per octave*[9] and *Hertz per Volt*[10] are two different methods used to control the pitch/frequency by step of an octave within an analog synthesizer. Manufacturers have used different control voltage range including -5 to 5v, 0 to 5v, 0 to 10v. These diverse implementations often lead to a difficult interoperability between modules. The first question that I was facing was concerning the needs to find a balance between cost, simplicity and quality. One of the simplest way of producing control voltage signals with a microcontroller is achieved using Pulse Width Modulation[11] (PWM). An output signal generated by the PWM pin of a micro-controller consists of an HIGH / LOW square wave signal whose width is modulated between 0 to 100% over time. The signal is generally smoothed out using a low pass filter and amplified to the Eurorack level. With this method one can produce an acceptable control voltage output. This technique is highly used on low cost Eurorack modules based on 8 bit microcontroller. More expansive circuits use a DAC chip for the digital to analog conversion, as result the output signal is often more precise. The quality and speed of the generated signal depend on its circuit design, combination of microcontroller + DAC chip and obviously the software implementation. *Duad's* design uses a 12 bit ADC and an external 16 bit DAC chip. A symmetric bipolar input/output range from

---

[6] **ARM**: *"previously **Advanced RISC Machine**, originally **Acorn RISC Machine**, is a family of reduced instruction set computing architectures for computer processors."* (Furber, S., 2000)

[7] **Arduino IDE:** an open-source software that simplifies the writing of the code and the communication with the board. The environment is written in Java and based on Processing and other open-source softwares.

[8] **SPI** Synchronous serial bus for communication to an external device (K66 Sub-Family Reference Manual)

[9] **Volts per octave:** ***"****one volt represents one octave."*

[10] **Hertz per Volt:** ***"****represents an octave of pitch by doubling voltage."*

[11] **PWM:** ***"****is a modulation technique used to encode a message into a pulsing signal."* (Wiki)

-5 to 5v has been chosen, this range is not ideal for a precise pitch control unit it rather opens a wider range of modulation possibilities.

Duad's aim is to share compositional ideas and other approaches to sound. Some of the details about its construction will be described in this thesis and should helps other users to use, extends or adapt *Duad* functionalities. The main reason that motivated me to build a new module has to do with limited resources and expensive standard market prices. A large number of modules are now available on the market, each with its own philosophy. Some of these modules are open-source and use an Arduino based approach that allows beginners, artists and performers to obtain a direct exploration of creative ideas. Other companies use a closed and optimized circuit design, that is not shared with the final users. Thanks to a wide open-source community *Duad* uses a number of libraries and open-source codes optimized for the Teensy. Numerous inspiring works related to modular synthesizer are openly shared and are available online. These projects help to spread knowledge about the solution of specific technical problems facilitating the development process.

2.3 Code and Libraries

Looking on the web for existing Eurorack modules based on the Teensy, I came across a post titled "teensy 3.x quad DAC board (Qu-ASR)" (Max Stadler) on Muffwiggler.com. The author of the post Max Stadler (mxmxmx) started publishing a series of open modules based on the Teensy. In particular Patrick Dowling, Max Stadler, and Tim Churches worked on a collaborative open-source project that ended up with the Ornament and Crime module (o_C): a Eurorack module entirely based on the Teensy 3.2, a polymorphic CV generator capable of generating four simultaneous voltage controlled outputs. The module was originally designed as a digital version of the Analog Shift Register (ASR). Several other apps have been added, it is now equipped with a collection of applications that explore a various and diverse approach to synthesis along with traditional functions such as LFO, quantizers and so on. These apps are selectable 'on-the-fly', without having to reboot the module or toggle the power. Several of the apps in the o_C re-propose and extend open-source code written by Oliver Gillet, the founder, owner, designer/engineer of Mutable Instruments, a Eurorack modules manufacturer. Oliver has publicly released on GitHub.com the firmwares of its modules collection along with an optimized set of classes, templates and utils written in C and C++ used for digital signal processing (DSP) on the ARM Cortex-M processor family. The Ornament and crime module represents a well conceived group collaboration that embraces many of my ideas. Thanks to these existing projects I was able to speed up the development of *Duad* and focusing more on the exploration of unconventional signal procedures. Below a list of the most important resources that relate to *Duad's* development:

Mutable instrument	Modules collection (Oliver Gillet) "https://github.com/pichenettes"
Ornament and crime	O_c (Max Stadler) "https://github.com/mxmxmx/O_C"
ADC	ADC library (Pedro Villanueva) "https://github.com/pedvide/ADC"
SPIN	SPI library (KurtE) **"**https://github.com/KurtE/SPIN"
ST7735_t3	TFT Display library (KurtE) "https://github.com/KurtE/ST7735_t3"
USB-Host	USB-Host library **(**Paul Stoffregen) "https://github.com/PaulStoffregen/USBHost_t36"

2.4 Hardware

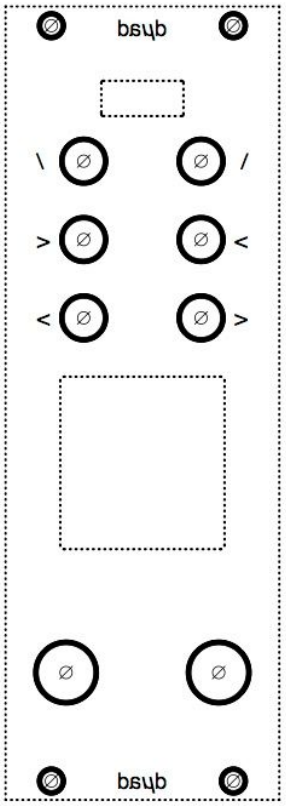


Figure 3: ***Duad*** front panel

Duad's front panel (Figure 3) follows a vertical symmetry to simplify its usability. **A** refers to the left side while **B** to the right side of the module. The process of defining the hardware of the module consisted on a detailed consideration of the input/output connections (I/O) available on the Teensy. i needs a system able to acquire two continuous analog signals used as control voltage inputs. Only two output channels have been considered during *Duad's* design. A small display is used to draw and adapt the graphical interface used to present different programs and controllable parameters. Two encoders with built-in switch buttons form the controllable hardware user interface. The Ornament and Crime base framework allows *Duad* to select which program is currently running.

The module uses: **\ /** as two digital inputs used as Trigger/Gate, **> <** as two CV bipolar inputs and **< >** as two CV bipolar outputs (see Figure 3). *Duad* is equipped with a Thin Film Transistor display 128x128 pixels (TFT), one of the most economic I have found. A female usb-port extend *Duad* functionalities, adding support for external controllers such as Mouse, Keyboard, Joystick, Midi and sensor devices such as the Arduino board.

Duad does not try to be extremely precise in terms of linearity and no digital compensation is applied to the output voltages. The linearity of the signal has not been considered as a priority assuming that *Duad's* practical use will be the one of a non-linear generator. *Duad*

has been designed specifically to produce slowly changing voltages suited to control various parameters of other modules such as frequency, resonance, amplitude and so on. The design of the module is focused on digital processing of control voltage signals, although its inputs can handle incoming signals with frequencies approaching the audible range with a maximum frequency of about 1.6 kHz. The main hardware of the module consists of a custom circuit board which essentially works as a dual control voltage breakout board for the Teensy microcontroller. A second circuit board hosts the user interface. Header pins are used to connect the two boards together completing *Duad's* circuit, see Figure 4. The 'homemade' front panel has been shaped out of a double layer copper 'empty circuit board' chosen as economic workable alternative to the aluminium and milled with a custom milling machine standing in my room. At this stage of prototyping most of the implementation choices have been dictated by economy.

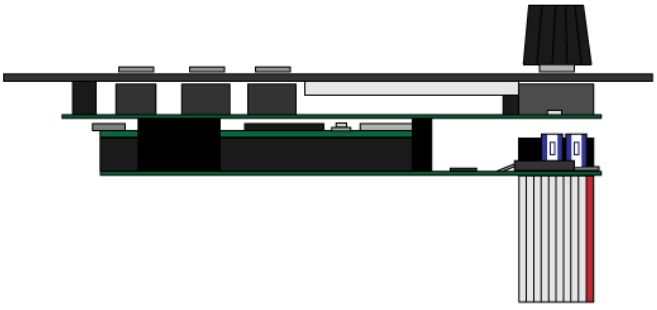


Figure 4: ***Duad*** sandwich

2.4.1 Circuit Layout

The main component of the board is the Teensy 3.6 micro-controller. Two on-board 12 bit ADC chips are available on the Teensy, namely ADC0 and ADC1; they can be read simultaneously. The board offers three parallel SPI buses SP0, SP1 and SP2 (K66 Sub-Family Reference Manual). The Serial Peripheral Interface Bus protocol (SPI) is a synchronous serial communication interface mainly used in short distance communication, primarily used to connect IC devices together. SPI0 is the main bus, it uses a four word FIFO (first in, first out) hardware queues that speeds up the transmission of data between the micro-controller and the DAC chip. The second SPI bus SPI1 is used to control the display, though it has only one word hardware FIFO queue. Not all the pins can be used to setup the SPI communications, the Teensy uses specific hardware pins for these buses. These communications choices have been influential and directly affecting the resulting circuits layouts.

The Teensy works at 3.3VDC and uses an internal voltage regulator to convert the standard 5V coming from the usb power cable (v-usb) to 3.3V level. A voltage regulator is used to convert the 12V coming from the Eurorack power rails into 5V required for the microcontroller. A couple of NPN transistors convert incoming Trigger/Gate signals within a threshold of ~2.5V into a HIGH/LOW (0V/3.3V) signal used by the digital pins of the Teensy. The board uses a double operational amplifier chip to convert two incoming bipolar signals

ranging from -5V and 5V into 0V and 3.3V respective signals used by the analog pins of the Teensy. The two outputs are generated by a 16 bit DAC chip PT8211 controlled by the Teensy using the SPI protocol. Generally the SPI requires a four wire connection, It is often called a four-wire serial bus. Here a basic example of Single Master to Single Slave connection:

Single Master to Single Slave example:

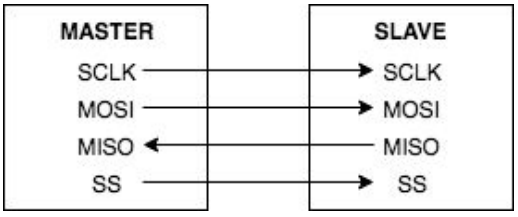


Figure 5: ***SPI*** - Single Master to Single Slave (Wiki)

SCLK: Serial Clock (clock output from master)
MOSI: Master Output Slave Input (data output from master)
MISO: Master Input Slave Output (data output from slave)
SS: Slave Select (output from master)

The Teensy is wired as master while the DAC IC is the slave. The DAC chip works as output device, so It does not need the MISO pin connection since it is not going to send any data through this pin. The 16 bit digital stream is converted into a respective analog signal in the range of 0.64V - 3.3V equal to about 2Vpp signal peak to peak. An offset and a gain factor of ~5V is applied to the signal using an inverting operational amplifier to produce an output voltage of about 10Vpp, the offset depends on the voltage reference that is applied into the non-inverting input of the amplifier, the range can be calibrated manually with a trimmer, the standard “calibrated” output voltage range is -5V to 5V, but can be set on the full positive range from 0V to 10V or in other particular configurations.

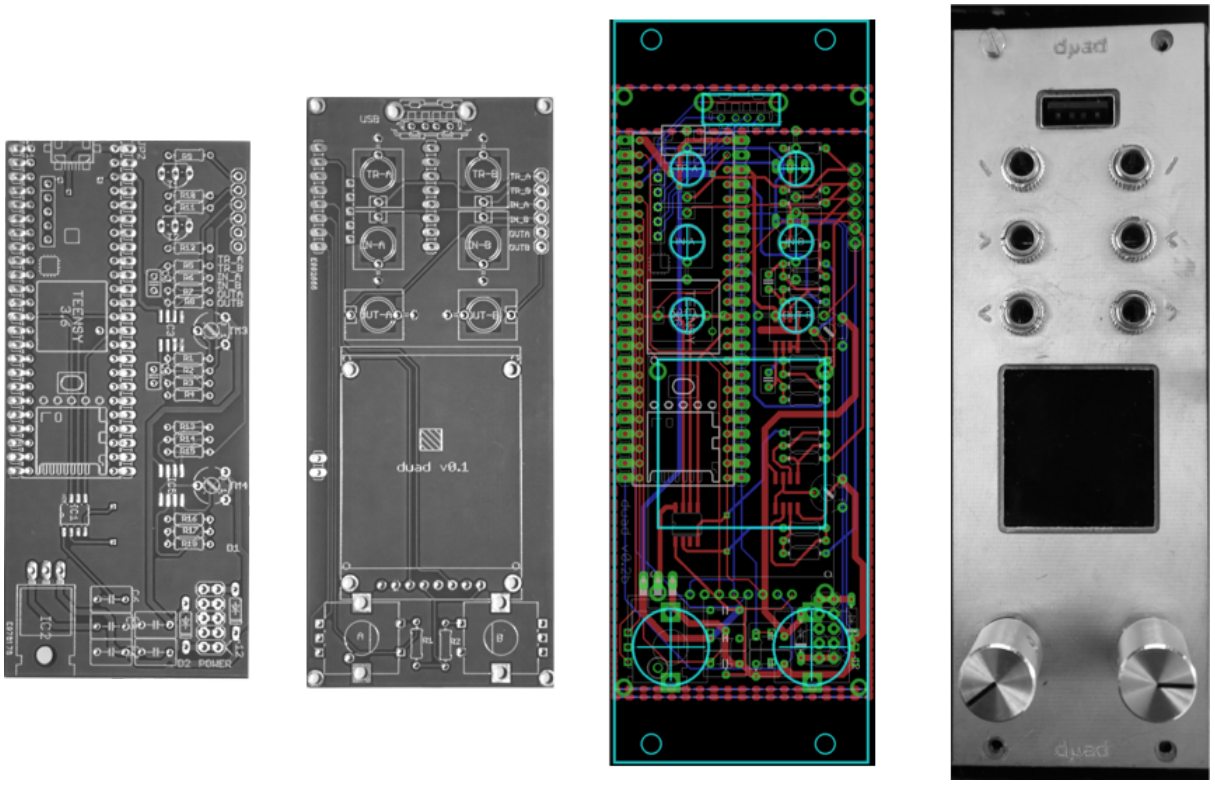


Figure 6: ***Duad*** circuits

2.5 Software

Avoiding to reinvent the wheel the code behind the module is highly inspired by many existing open-source Eurorack projects. The code is a mixture of Inline assembly, C and C++ languages and mainly based on open-source codes written for the Ornament and crime and Mutable instrument modules. The code consists of an Arduino IDE file project called *duad*.ino. C structures and C++ classes are used to organize and isolate the various responsibility of runtime operations. The project files are organized into a single folder called ***duad***:

/duad/
- duad.ino
- config.h
- gpio.h
- duad_adc.h
- duad_adc.cpp
- duad_dac.h
- duad_dac.cpp
- duad_digital.h
- duad_digital.cpp
- util.h
- ….

An Arduino Sketch consist of two main function called setup() and loop(). The setup() function runs only once, It generally contains some initialization statements that only need to run once. The loop() function is the main loop of the program and run repetitively along its life cycle. This loop consists of a top to bottom execution of operations, when the bottom of the loop function is reached the loop begins again.

```
//Example.ino
void setup(){
        //Initialization code run once
}

void loop(){
        //begin loop
        Operation 1;
        Operation 2;
        Operation 3;
        //end loop
}
```

2.5.1 Control flow

Data flow within a digital system occurs at different rates depending upon the production and consumption of the data.

Some typical examples of data rates needed in *Duad* are:

- Rate at which new values are received from an hardware rotary control when it is varied;
- Rate at which a graphical display is refreshed to show the current amplitude level
- Rate at which audio samples are received from or sent to a digital/analog interface.

To organize and divide priorities between various tasks of a microcontroller, a digital clock or timer is commonly used. A master-clock is the one that runs at the maximum speed. Other speeds are then obtained by subdivisions of the master-clock reaching slower execution rates. *Duad's* firmware uses an internal timer that run at 16.67 kH, corresponding to 60 microsecond per cycle of the main Interrupt service routine[12] (ISR). *Duad* has not been designed as audio generator and uses a slower sample rate to obtain time frames suitable for the control of sound.

The following pseudo-code describes the basic application flow:

```
//Main.ino
void ISR(){
        // Read ADC
        // Write DAC
}

void setup(){
        // Initialization ADC and DAC
        // Initialization of Timer that trigger the ISR() function at 16.67 kH
}

void loop(){
        // Read encoders position and buttons state
        // Change application running
}
```

[12] **Interrupt service routine** "is a software process invoked by an interrupt request from a hardware device. It handles the request and sends it to the CPU, interrupting the active process. When the ISR is complete, the process is resumed." (Christensson, Per. "ISR Definition." TechTerms. (December 7, 2016). https://techterms.com/definition/isr

2.5.2 Duad programs

Duad firmware implements a basic programs menu that allows to change the running program without having to reboot the entire module. The menu is accessible by a long-press on the right encoder which is also used to scroll and select the running program.

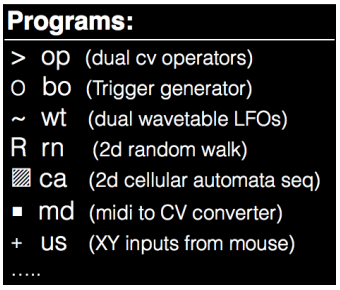


Figure 7: ***Duad*** - Menu

A program consist of a C++ class that define and implement a specific abstraction. The available programs can be divided in two categories: *single* and *dual*[13]:

Single programs are subprograms of the Split functionality (see chapter 2.8). A subprogram abstracts an idea considering only one half of the module (see Figure 11). These type of programs use one incoming signal and produce one single output signal. Sample and Hold, Low frequency Oscillator, Rungler, Clock Divider are just few example of the subprograms implemented in *Duad*.

Dual programs are complete application that exploit the full potential of the module. These programs are capable of processing in parallel two incoming signals producing two distinct outputs. External digital interfaces such as Midi controllers, mouse and keyboards can be used to interact with a particular program. Even though several are the programs that have been written for *Duad* (such as 2d Random walk, 2d Cellular Automata, Midi to CV, Dual LFO, Mouse to CV etc..) only two of them will be analysed in details through this chapter.

[13] Inside the source folder '/Duad/res/' two scripts have been made to facilitate the creation of an empty program.

Terminal.app (MacOs):

./new_single.sh "PROGRAM_NAME"

./new_dual.sh "PROGRAM_NAME"

2.6 Bo

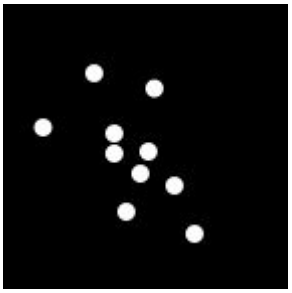


Figure 8: ***Duad*** - Bo program interface

Bo is a bi-dimensional particle system that simulates the behavior of circular particles within a “virtual world”. A simple collision detection system has been implemented to detect and react at each particles’ collision producing generative and complex streams of pulses. A notable inspiration is the *OP-1* synthesizer (from Teenage engineering) where a similar functionality called “Tombola” is used to trigger various synthesis engines. A number of previous experiments have been considered during the development of Bo. Some of my previous works, presented during my studies at Sonology (Appendix A: Collision Networks) were already using various implementations of interactive particles systems in combination with Supercollider to produce rich dynamic interactions usually mapped in the form of controls signals for synthesis parameters. Bo instead is used to control analog patches where complex series of pulses are desired. This dynamic simulation can vary from an elastic to an inelastic system whose dumping/friction and gravity are controllable by the user. The boundaries of the display define the edges of the map. Particles are attracted to the center of the map by a gravity factor.

A curious aspect that affects Bo simulation is the so called Zeno paradox: “*It can be argued that Zeno behavior is little more than a mathematical curiosity and should not be an issue when dealing with physical systems. However, modeling abstraction, often employed by engineers to simplify models for the purpose of analysis and control, can easily lead to Zeno behavior,…When faced with Zeno executions, simulation algorithms are likely to stall or produce incorrect and unpredictable results...Zeno's motion paradoxes all have to do with dividing time into ever smaller slices...The idea that the ball executes an infinite number of bounces in a finite amount of time is perhaps the most interesting response to Zeno's paradoxes.*” (Millersville University). Despite Bo does not try to particularly emphasize Zeno’s behaviors, there are various practical cases in which particles can start to behave in wrong but interesting ways. Bo simulation can exhibit undefined and unpredictable behaviours; with a wrong configuration of parameters balls start to intersect each others allowing the system to behave in unstable manners. Two or more balls can partially or completely overlap in undefined situations (see figure 9b) that result in chaotic and repetitive series of pulses. These strange situations are exploited as they produce interesting dynamic reactions useful in generating compex control structures. Some of the most intriguing aspects of these structures are being implicitly chaotic as well as systematic.

2.6.1 Implementation

Bo is an idealized model of the motion of atoms or molecules in a container. N particles in motion are confined in a screen. Particles have known position, velocity, mass, and radius.
A variable force attracts the particles at the center of the container allowing them to trace orbital motions. The attraction point has been placed at the center of the screen in order to maximize the potential orbital areas. Particles interact via collisions with each other and with the reflecting boundary generating a dynamic stream of pulses.

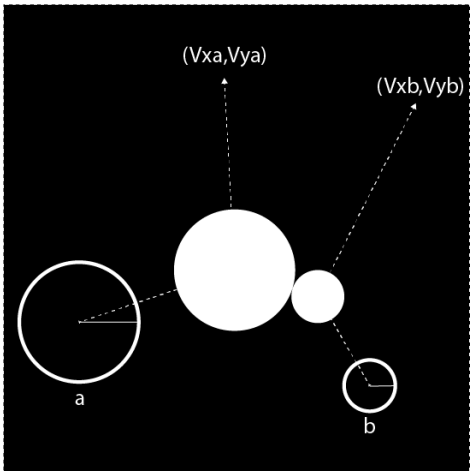


Figure 9a: defined particle collision

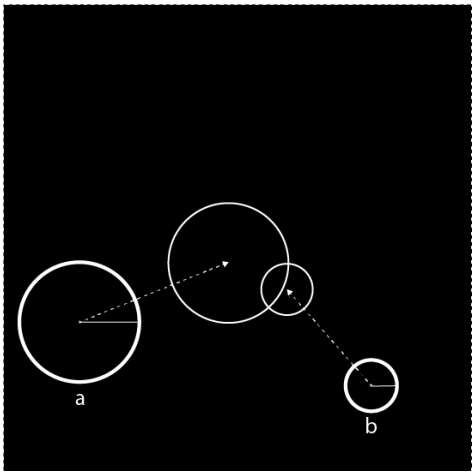


Figure 9b: undefined particle collision

The particle system can be controlled externally, parameters such as friction and attraction can be modulated by external signals or set to a constant value using the respective encoders (see Figure 10).

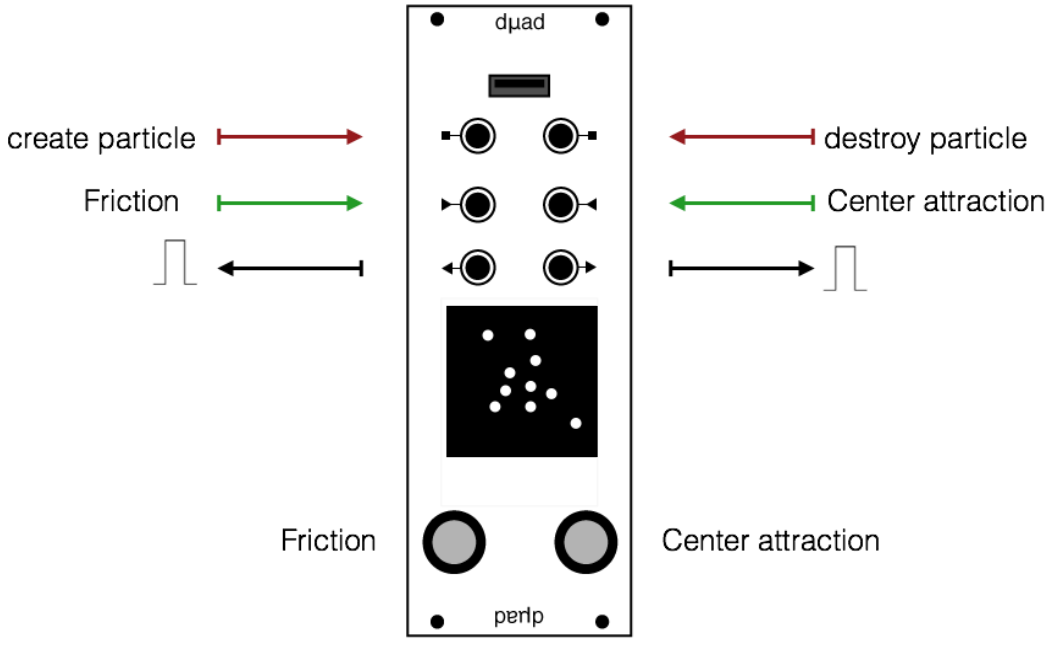


Figure 10: Bo - parameters

2.7 Op

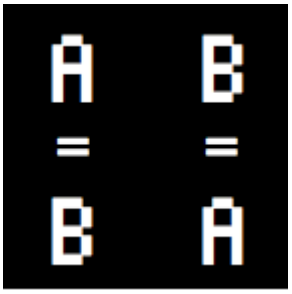


Figure 11: ***Duad*** - Op program interface

"*The general signal processing problem is to take a vector x as input and return a vector y as output, where y is related to x according to some rule.*" (Manton, J.H).

Op is the first and probably the most simple program I wrote for *Duad*. In its basic form It consists of a programmable comparator capable of comparing simultaneously two incoming analog signals deriving from the relations between the inputs. The design of OP is similar to the one of an Operational amplifier[14] (Op-Amp). It extends the idea behind the variable level comparator module (VLC) available in Bea 5 studio at the Institute of Sonology in The Hague. The VLC has two inputs and two outputs. It compares the two incoming voltages and always selects the higher of the two on output one, and the lower on output two. The other output, called 'comp', is 5 volts when A is higher than B, and 0 volts when B is higher than A. Even if this device was designed to function in the low frequency range, it can also be used to produce complex waveforms in the audible range.

Inputs:	A	(V)
	B	(V)
Outputs:	high	(voltage)
	Low	(voltage)
	Compare	(voltage)

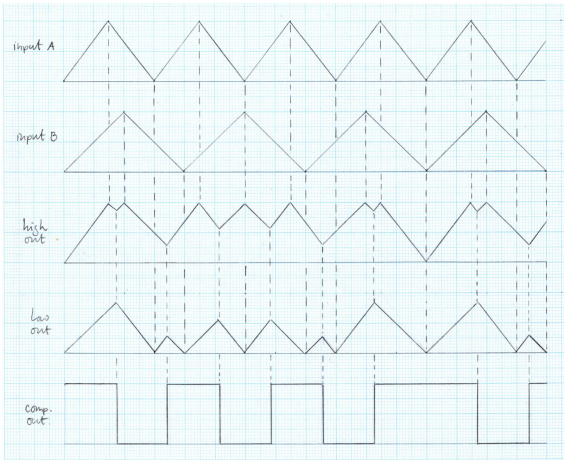


Figure 12: VLC - Studio manual BEA5, (2009)

[14] **Operational Amplifier**, or Opamp is a DC coupled high-gain electronic voltage amplifier with a differential input and, usually, a single ended output; (Wiki)

The concept behind Op is very simple. It has been inspired by various existing instruments and ideas. The basic idea comes from Supercollider Ugen implementation, where binary operators can be easily used to manipulate sound processes (Koutsomichalis, 2013, p. 39) . Another example of low-level computer operations used in a musical contexts are the previously mentioned ByteBeat formulas (see chapter 1.3) where the phase or time variable is incremented and recursively processed by means of arithmetic and binary manipulations and used to create a continuous audio waveform. Another inspiring work is the CrackleBox (Waisvisz, 2004), an instrument developed at Steim. Michel Waisvisz designed and built the very first Crackle circuit in the late 60's together with Geert Hamelberg. It works as a musical instrument whose circuit consists of a single operational amplifier chip directly controlled by the physical interactions of the player. The instrument is manipulated by open circuit points touched by the fingers. The player's capacitance and resistance are physically mapped and used to connect various parts of an unstable oscillating circuit, creating undefined and noisy musical structures. Op allows to shape control voltage signals that evolve in time depending on the current state of the program. Two slow incoming signals are repeatedly confronted producing new unexpected waveforms.

2.7.1 Implementation

Based on an analog circuit implementation the VLC module produce two respective outputs signals that depend on a fixed design. Op's operations are not limited by the circuit and can be dynamically altered during the execution of the program. Op uses 32 operators, generally used in computer programming to manipulate numbers. These computer operators define the state of the unit. Each state derives from the relations between two incoming control signals. Op implementation is based on purely digital operations common to the analog world such as sum (+) , subtraction (-), multiplication (*) etc.., besides more interesting and unpredictable results can be achieved by using binary operators. These relations are represented as a set of basic mathematical equations: OUT = A **op** B where **op** can be selected by the user. The program uses the common C programming language operators set as instructions to process and calculate the output signals. The set includes arithmetic, bitwise, conditional and logic operators. Here a list of all the available operators:

Arithmetic operators:
addition, subtraction +, -, ++, --
multiplication, division, modulus *, /, %
Comparison operators:
less/greater than (or equal to) <, <=, >, >=
(not) equal to ==, !=
Logical operators:
Not, logical AND, logical OR !, &&, ||
Bitwise operator:
bitwise complement ~
bit shift left/right <<, >>
bitwise AND &
bitwise exclusive OR ^
bitwise inclusive OR |
Compound assignment operators:
+=, -=, *=, /=, %=, |=, ^=, <<=, >>=

Op graphical interface has the form and syntax of two parallel mathematical equations whose default operator is the equal symbol (=). The left side of the program states A = B , while the right side B = A. A and B are respectively the Left and the Right available inputs. The calculations result in two outputs signals, namely A and B. In practice Op is a symmetric one-instruction program that can be 'reprogrammed' in real-time. The possibility of alternating between different states makes Op capable of exploring a wide range of behaviours that characterises the control of sound generating processes. Op can be used in various configurations within different modular patches and has been mainly designed to help in the exploration of dynamic hybrid patches. Op has been inspired by the general concept of a Finite State Machine[15] (FSM). Each operator in Op corresponds to possible states or signals transformations. Different operators can be set manually with the respective encoder, or can be controlled externally with a trigger. (see Figure 13)

When using only square waves in both inputs (><) with same amplitude levels, no matter what is the operator selected, the resulting output signal will alternate between zero, minimum or maximum voltage without any values in between. By varying the amplitude or phase of the two inputs it is possible to obtain rather useful rhythmical structures so crucially affecting the melodical identity of a piece. Using triangle, sawtooth or sine as waveforms the resulting outputs will often contains varying waveforms emerging from the relations of the inputs signals. All operations are executed at a sample rate of 16.67kh inside the ISR function of the running program. The samples calculations use incoming amplitude values with 12 bit resolution. The C code uses a 16 bit integer datatype to describe an individual output voltage sample. Op, like all the applications running on *Duad* has been implemented using integer arithmetic to speed up its execution.

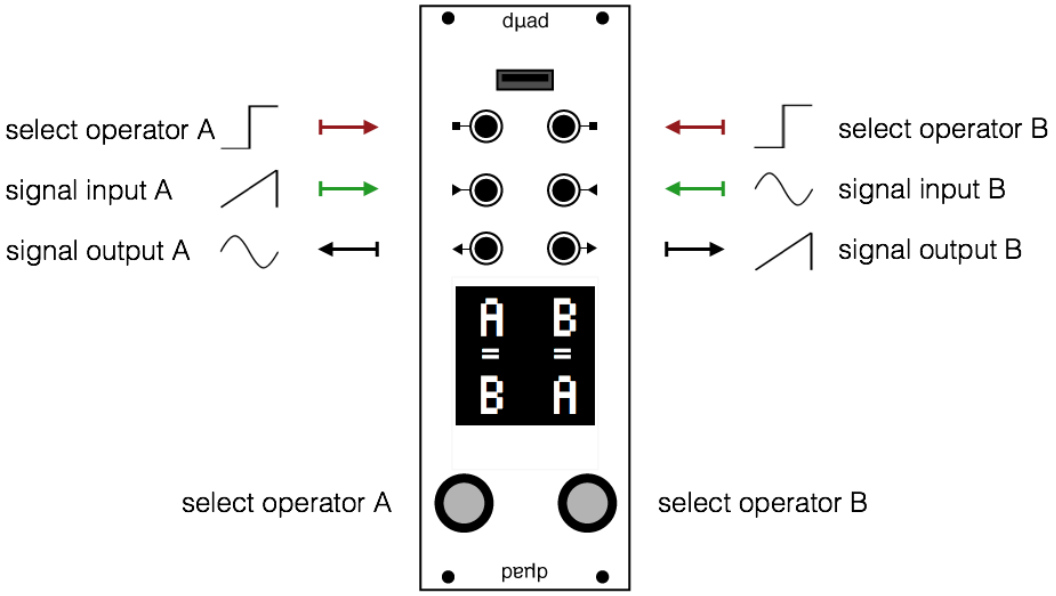


Figure 13: Op - parameters

[15] *"The standard **finite state machine** (fsm) contains a finite number of states, and a transition or next-state function that maps states and events (inputs) into states. At any point, the system is in a given state. The occurrence of an event causes the system to change state according to the transition function definition."* (Shaw, A. 1998)

2.8 Split

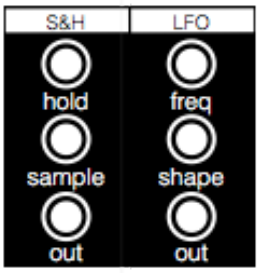


Figure 14: ***Duad*** - Split program interface

Another interesting program that is worth mentioning is **Split**: Split has the ability to run independently two parallel functionalities, splitting the module vertically in two parts. Each part can run different subprograms (i.e. Sample and Hold, LFO, Rungler, etc..).

Duad firmware facilitates the implementation of new programs. Compact pieces of code define the subprograms available in Split and are collected in a single file called 'Module.h'. New ideas can be implemented allowing other musicians to create or customise a program for a specific performance or a sound installation.

2.9 Reflections

Duad opens up a wide range of control signal manipulations, moreover its uses can apply to installations where sensors or actuators are mapped to sounds. Future developments of *Duad* will consist on a slow and precise fine tuning of the code, the available programs will be extended along with bug-fix and other general improvements. *Duad* helped me in a deep understanding of what control voltage can offer to the modern composer. A non-standard approach to sound manipulation has been explored introducing new abstractions during the production of a sound composition. Being able to program and test 'on the fly' a new idea, as it happened with Op, into a modular patch allows me to think about new and possibly interesting forms of control that could extend my compositional possibilities. Perhaps this approach, characterized by a 'mutable model' that can be altered in real-time, is one of the central ideas of *Duad.* By rapidly changing the currently selected program (or the operator in Op), the module start to generate truncated modulations patterns and undefined situations resulting in complex and dynamic evolutions of control signals. *Duad's* functionalities have not been yet completely exploited, its full potential still needs to be explored in various musical situations.

Chapter 3

Sound results

I often find difficult to define the boundaries between musical composition and sound experimentation. My music does not necessarily focus on a single idea, It rather explores various parameters spaces in which form or structure can be created. More extensively my investigation regards the observation of generative sound processes and all the collateral and undefined circumstances inevitably attached to them. Dram and Yyyyttt are just two among other fixed-media pieces that explore different approaches to sound composition, In this chapter I will describe the development behind these.

3.1 Dram

Dram has been composed in 2017 during Composition workshop, class lead by Kees Tazelaar. The structure of the piece has been inspired by the framework conceived by M.G. Koenig for the composition of Terminus 1.

In Terminus 1 Koenig developed a system in which sound materials are divided in individual blocks (or phrases). These blocks are then transformed by means of recursive analog manipulations in the analog studio. The results of these transformations can be represented by a branched tree that grows in vertical and horizontal directions. At each iteration the sound result is processed again advancing the tree structure, moving away from the original material.

"The form of Terminus 1 is based on the fact that transformations of the sound material which are themselves already transformations become less and less related to the original source material." (Koenig 1971, Ästhetische Praxis p. 103)

Dram's starting materials consisting of short sound recordings of percussive instruments such as drum kit, wooden and metallic objects. The sound is processed by means of recursive transformations. The original sound recordings have a duration ranging between one and three seconds and needs to be phrased in some way. The idea was to create longer rhythmical patterns by combining different sound characters together. Supercollider has been used to generate longer patterns that serve as building block for the piece, in particular a granular technique is used to blend two rhythmical characteristics together. This granular process uses a probabilistic distribution that defines the transition between two grains. I was inspired by a snippet of Supercollider code from Nick collins' online tutorials where he describes a similar probabilistic procedure: "*Granular crossfade; as increase density of grains from one, reduce from another, done in a basic probabilistic way*." (Collins Nick). This method appears to be effective with percussive sound materials and it has been intensively used to create long percussive sequences that combine diverse sound

characters. These transformations effectively mix two diverse percussive sounds allowing to create extended rhythmical patterns that change over time. The materials have been transformed several times, using the same aforementioned granular technique. Numerous variations of these materials have been recorded with different playback-rate (speed), grain size and probability distribution. The resulting materials have been manually arranged in time to form the final piece. The piece lasts 12 minutes during which percussive structures are combined to form a single rhythmical stream. The composition traverses only one of the possible trajectories allowed by the framework, that is capable of generating almost infinite alternative versions from the same original sound material. The generative aspect of this framework is still appealing for contemporary music composition. It allows me to map and access each step of a recursive process, where steps may bifurcate into branches enlarging the space of the exploration. The spatiality of the sound was not particularly considered during the development of the piece and was composed in a stereo format. Numerous tests regarding its spatialization have been made lately, as result the sound material appears to be more consistent when reproduced in a stereo configuration.

3.2 Yyyyttt

Yyyyttt is a sound composition made-in-duad. The piece started from a collection of sound recordings made during the development and testing phase of the Op program. About few minutes of audio were captured at each test where interesting behaviors have been recorded. Fast and slow evolving modulation patterns that sound like ‘animals’ and other sounds that resemble the sound of a Dial-up Modem or a direct binary audification form the basic sound materials of the piece.

The composition consists of five layers of materials that have been produced with shared or similar modulation settings between various iterations of Op. Rhythmics patterns that slowly evolve in time arise from the relations between two slow analog signals. Each layer occupies a specific band on the frequency spectrum allowing various voices to coexist. These bands have been chosen empirically based on the quality of the sound materials. The patch consists of an analog filter used as sound generator in combination with a dual LFO module fed into Op’s inputs. The frequency and resonance of the filter are modulated by signals resulting from Op operations. The two operators are self modulated by feedback loops within the system. A spring reverb has been placed at the end of the audio chain and used to smooth some of the sharp sounds. The results of Op have been scaled within different ranges. In the first and second layers Op controls the full frequency range of the filter resulting in wide and expressive pitched gestures. The others layers use restricted frequency bands whose center frequencies are slowly modulated over time. These layers alternate between voiced and unvoiced sounds. The piece results in a dense ecosystem where various sounds are trying to coexist affecting each others. Op has been useful in generating a large number of unstable and undefined behaviours often producing broken or malformed modulation patterns which are then used to define an individual sound event. (see Figures 15-16) Yyyyttt is just one of the possible pieces that can be made with this material, the sounds produced alternate between percussive and pitched sound surrounded by noise and unpredictable situations.

Here some examples of patches that have been used in **Yyyyttt :** (depending on the LFOs amplitude values, phase differences, the filter notch-balance and the current operators, strange modulations start to emerge)

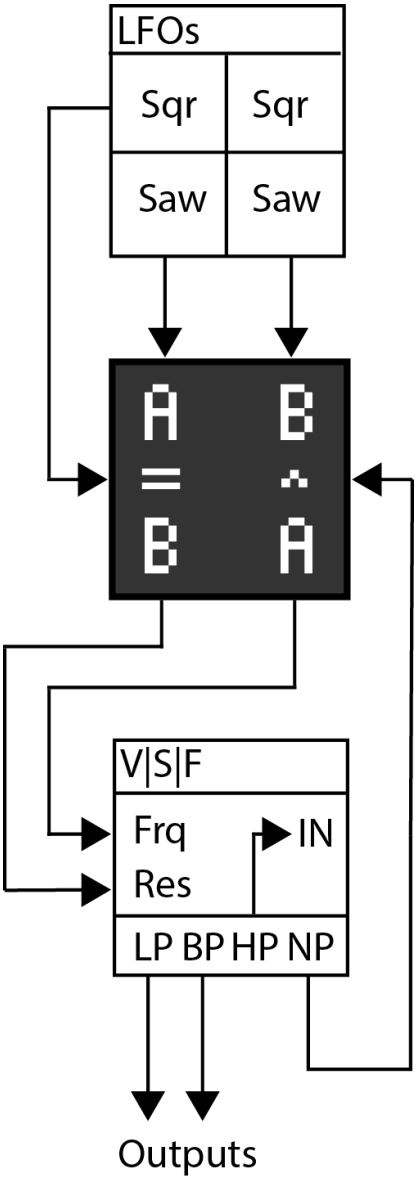


Figure 15a: ***Yyyyttt*** - patch **A**

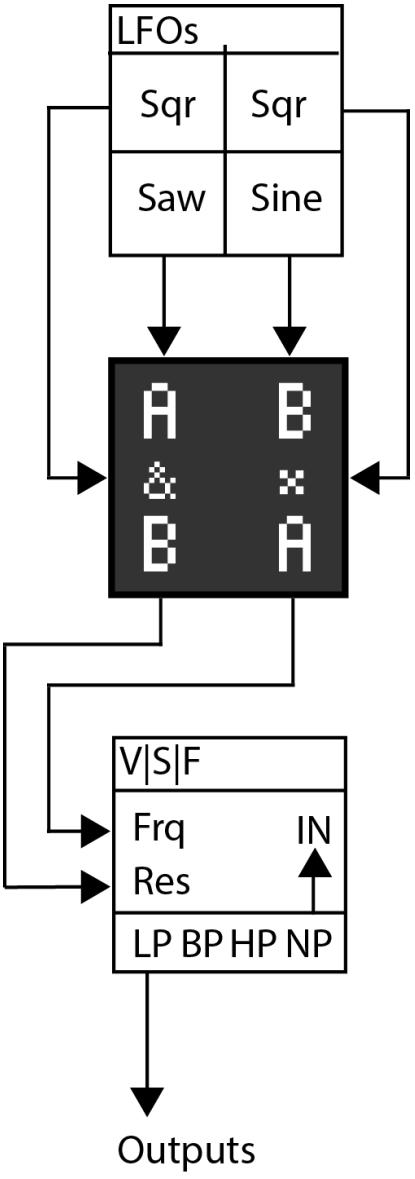


Figure 16a: ***Yyyyttt*** - patch **B**

Chapter 4

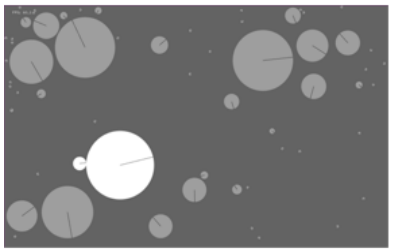
Conclusion

Many thoughts concerning sound have been made during the last four years resulting in an exploration of generative and dynamic sound possibilities. The research presented practical examples of abstraction and their possible musical uses. An open-source versatile hybrid module called *Duad* has been fabricated, along with a number of sound compositions and possibly infinite sound control generators. *Duad* will still keep me busy for quite some time. The research led me to experiment and learn various non-standard synthesis techniques and compositional ideas that enlarged my view on sound and its organization in time. My own artistic works will continue to explore sound and its derivatives. Despite this research has tried to open other perspectives on the control of audio parameters, abstract models are only a tool that allows the composer to define and limit compositional parameters otherwise difficult to organize. What the composer does with those models is the most significant aspect of the work. While dealing with an infinite set of possibilities offered by the computer the difficulty is to find a balance between the abstract idea and a concrete procedure. This research proposed a physical exploration (made with patch cables) that allows me to directly interact with those parameters and mapping strategies. By restricting the scope of the research and focusing on a small modular synthesizer I was able to test ideas on a relatively simple musical instrument still capable of interesting sound results. The modular aspect of this type of synthesizers allows for further investigations where new combinations of modules will certainly extend the sound and compositional approaches.

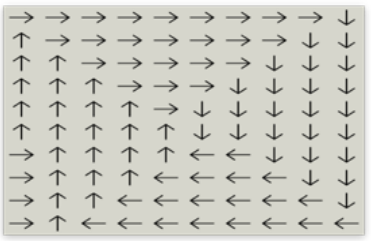
Appendices

Appendix A

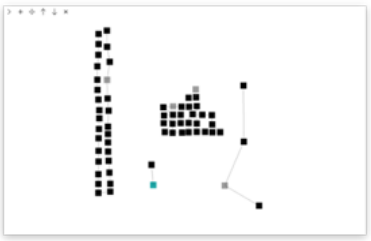
Osc experiments



(Collision Network)



(Seq, directional sequencer)

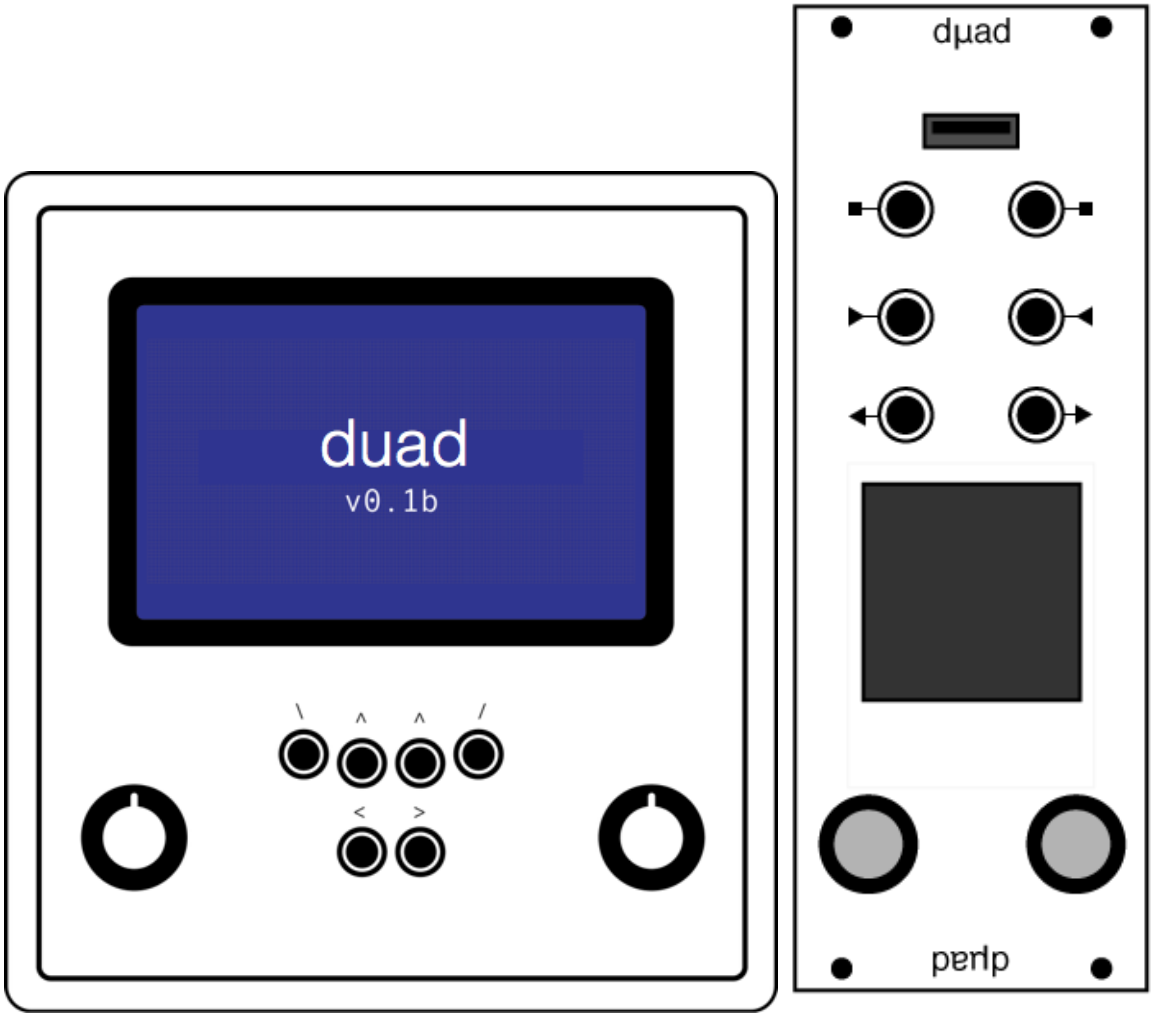


(SoQ, Osc events sketch)

Appendix B

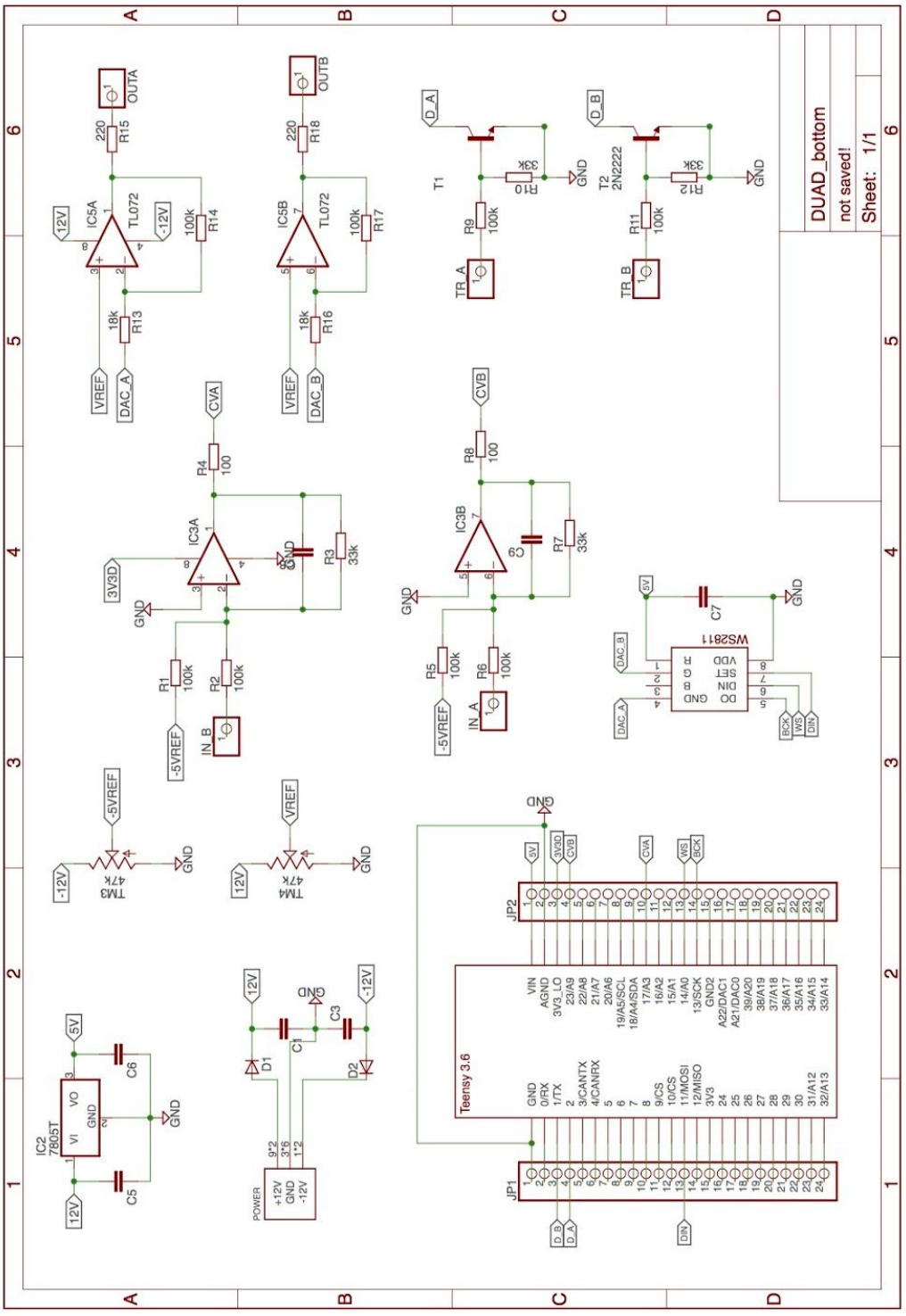
First prototype of Duad (v0.1b)

Last prototype of Duad(v0.3b)



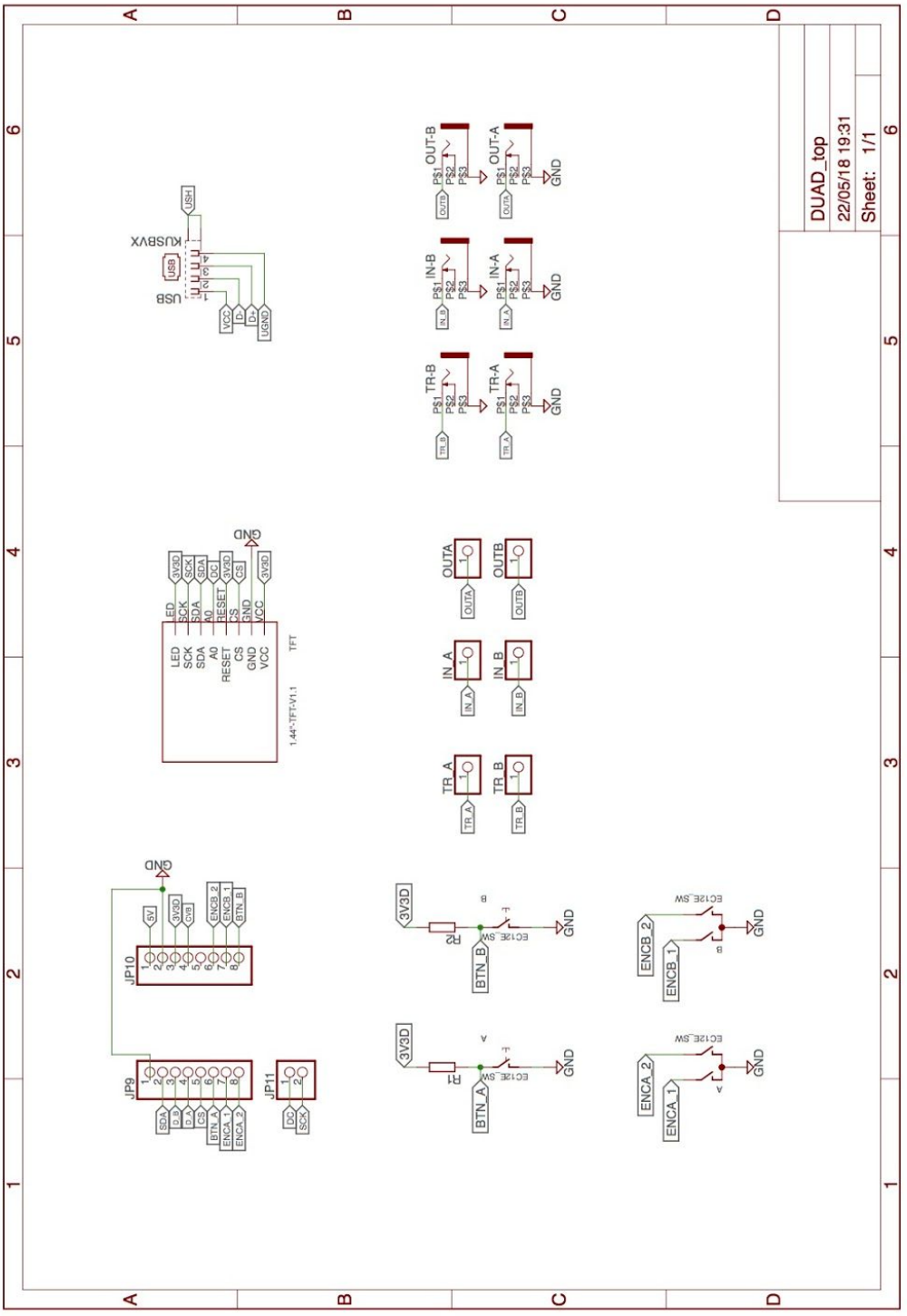
Appendix C

Duad bottom schematic (Eagle CAD)



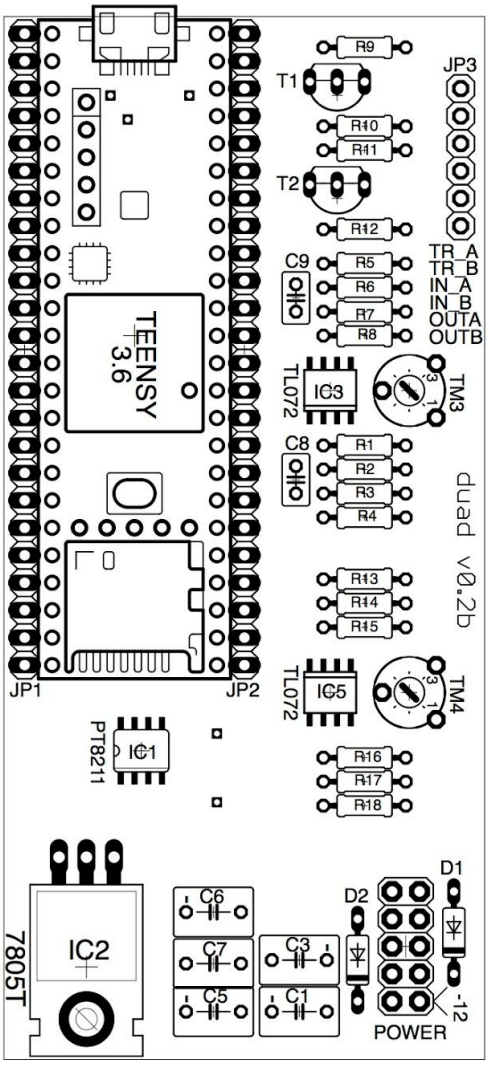
Appendix D

Duad top schematic (Eagle CAD)

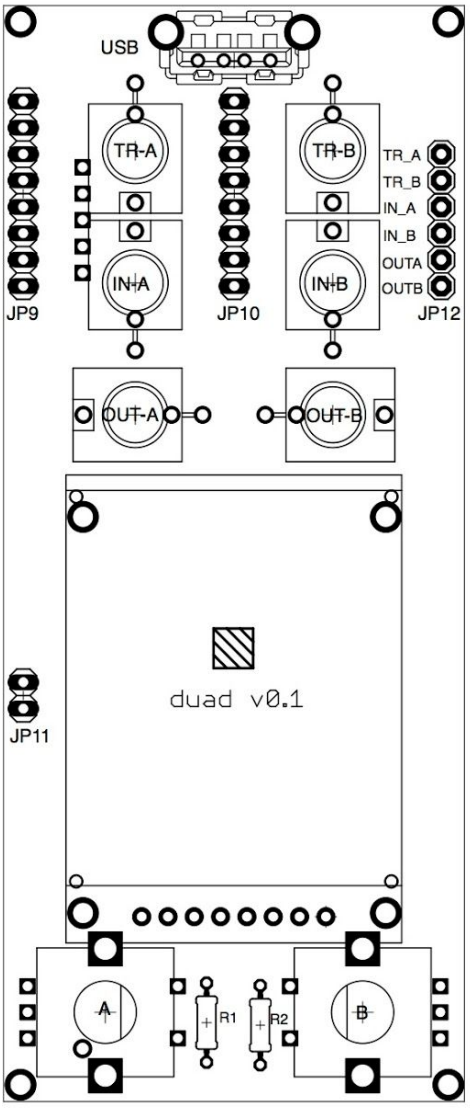


Appendix E

Duad bottom PCB (Eagle CAD)

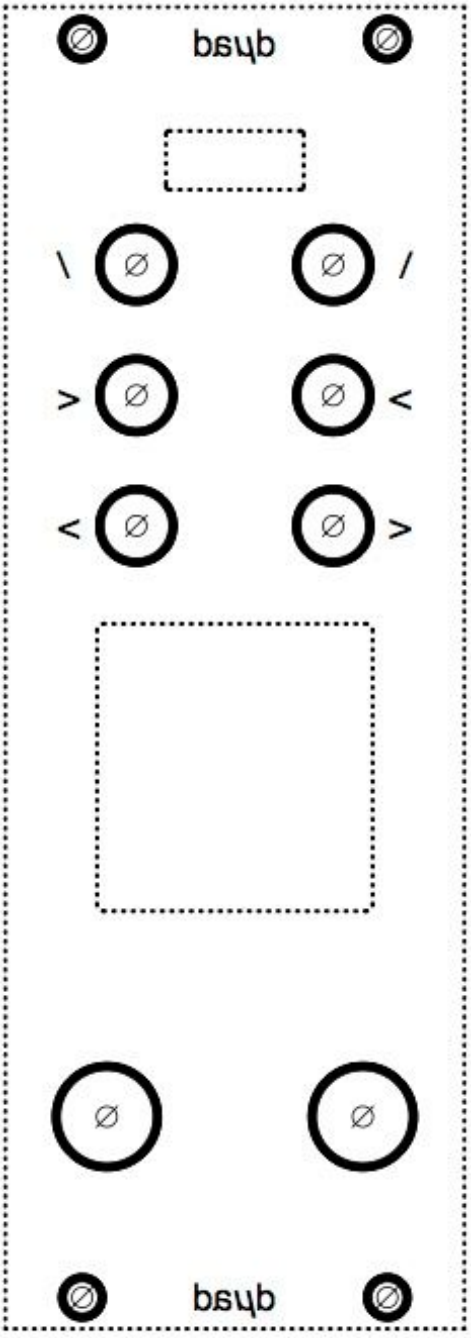


Duad top PCB (Eagle CAD)



Appendix F

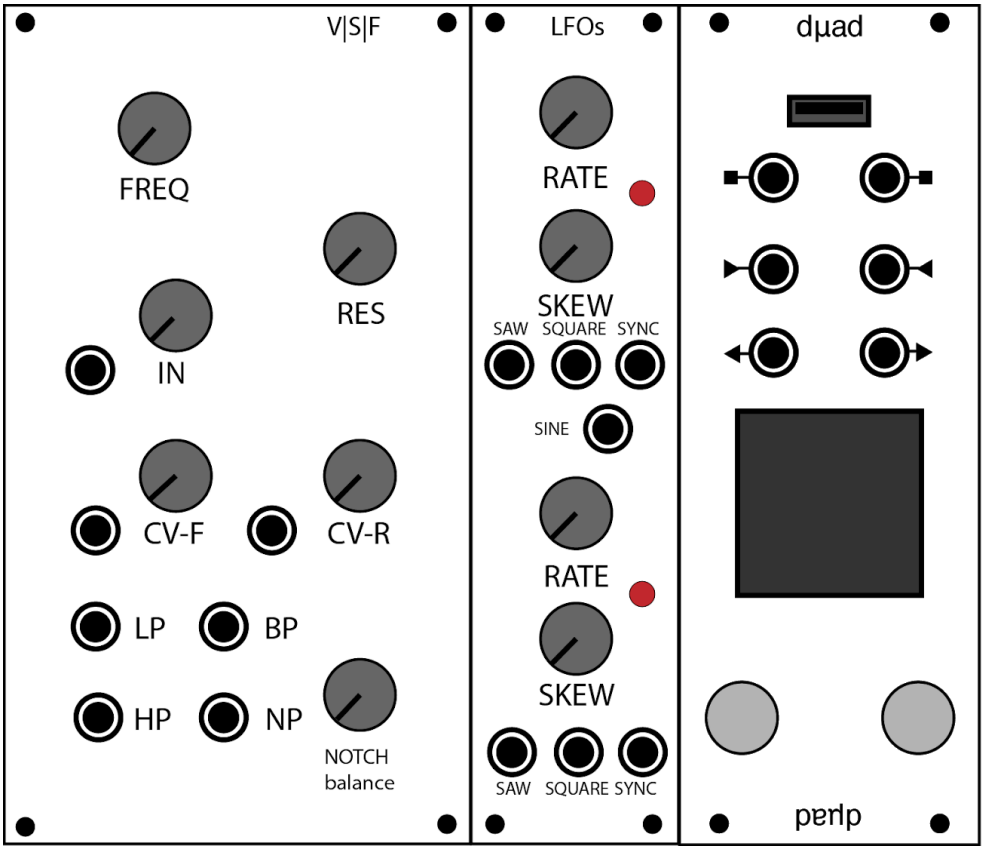
Duad front pannel (Eagle CAD)



Appendix G

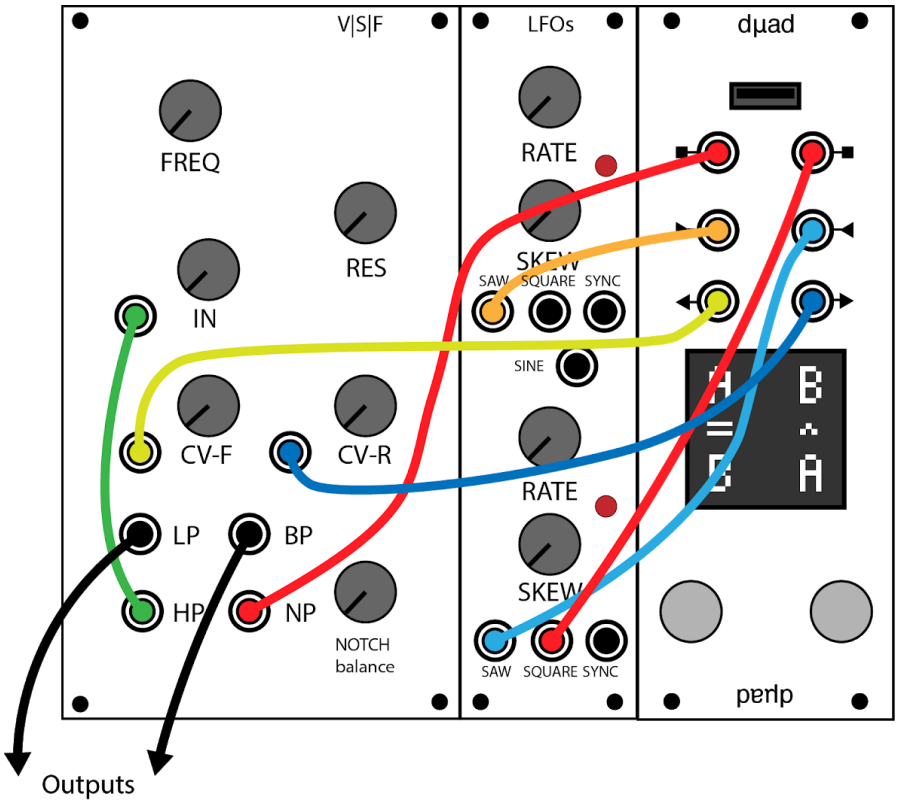
Modular setup

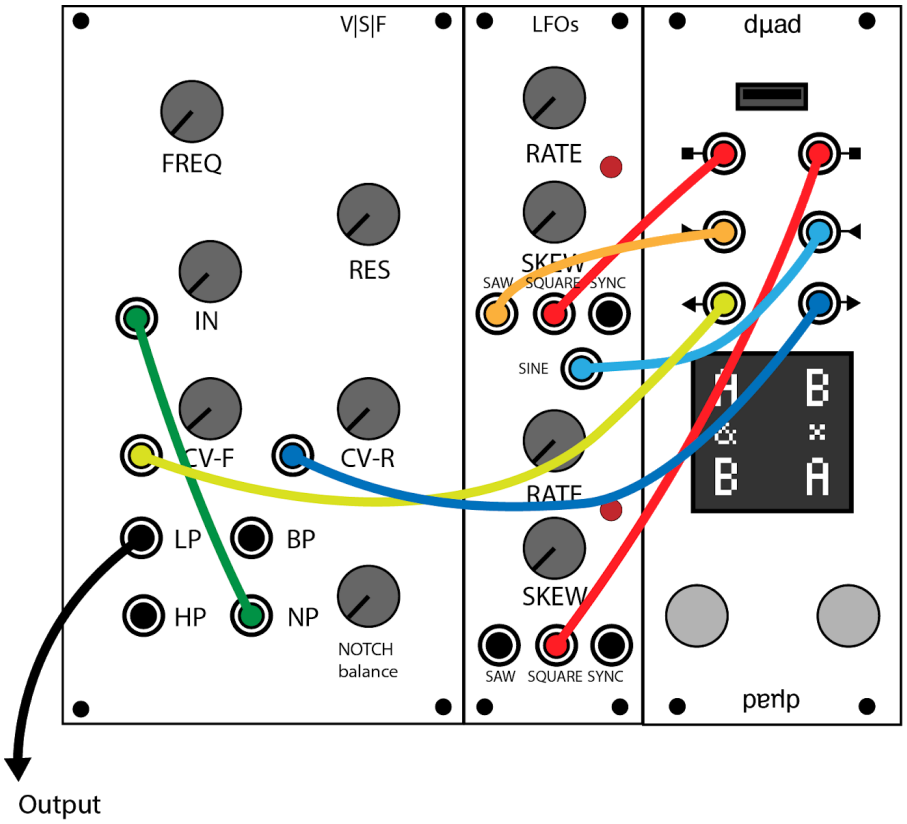
{*Variable State Filter , Dual LFO , Duad* }



Appendix H

Yyyyttt patches A and B





References

Priemer. R. 1991. *"Introductory Signal Processing."* World Scientific

Koutsomichalis, M. 2013. *"Mapping and Visualization with SuperCollider."* 34 - 40.

Bellomo, N. Preziosi, L. 1994 *"Modelling Mathematical Methods and Scientific Computation."*

Heikkila, VM. 2011a. *"Experimental music from very short C programs."* YouTube video, http://www.youtube.com/watch?v=GtQdIYUtAHg

Heikkila, VM. 2011b. *"Discovering novel computer music techniques by exploring the space of short computer programs."*, https://arxiv.org/pdf/1112.1368.pdf.

Heikkila, VM. *"Bringing magic back to technology",*
http://viznut.fi/texts-en/magic_back_to_technology.html

Delton, T. Horn. 1984. *"Music Synthesizers - A Manual of Design and Construction"*

Guttag, J.V. 2013. *"Introduction to Computation and Programming using Python"*

Holtzman, S.R. 1979. *"A description of an automated digital sound synthesis instrument."* Computer Music Journal, 3(2).

Koenig, G.M. 1971. *"The Use of Computer Programmes in Creating Music.",*
http://www.koenigproject.nl/Computer_in_Creating_Music.pdf

Berg, P. 1979. *"PILE: A Language for Sound Synthesis."* Computer Music Journal 3(1): 30 - 37. Revised version included in Roads 1985.

Berg, P. 2009. *"Composing sound structure with rules"*, Contemporary Music Review, 28: 1, 75 - 87.

Döbereiner, L. 2009 *"Compositionally Motivated Sound Synthesis"*,
http://doebereiner.org/texts/doebereiner_ngen.pdf

Roads, C. 1978. *"Interview with Gottfried Michael Koenig."* Computer Music Journal 2(2): 62

Heath, Steve. 2003. "*Embedded systems design*." EDN series for design engineers (2 ed.). Newnes. pp. 11–12.

Furber, S., 2000. *"ARM System-on-Chip Architecture."* Addison Wesley

Hordijk, Rob. *"Designing Instruments for Electronic Music."*

Max Stadler (mxmxmx) *"teensy 3.x quad DAC board (Qu-ASR)"* ,
https://www.muffwiggler.com/forum/viewtopic.php?t=95604

Freescale Semiconductor, Inc. Rev. 2, May 2015. *"K66 Sub-Family Reference Manual",*
https://www.pjrc.com/teensy/K66P144M180SF5RMV2.pdf

Millersville University, *"Experiment of the month - Zeno and the bouncing ball",*
http://www.millersville.edu/physics/experiments/045/index.php

Waisvisz, Michel. 2004 *"Crackle History"*,
http://www.crackle.org/CrackleBox.htm

Manton, J.H. *"Differential and Algebraic Geometry in Signal Processing"*,
https://people.eng.unimelb.edu.au/jmanton/pdf/Manton_S_math_in_sp.pdf

Shaw, A. 1998. *"State machines"*,
https://courses.cs.washington.edu/courses/cse403/98sp/notes/state-machines.html

Teenage engineering, *"Op-1"*.
https://www.teenageengineering.com/products/op-1

Collins, Nick. *"Granular Bonus examples"*, 5. Sound Synthesis 2: Sample-based, Granular
https://composerprogrammer.com/teaching/supercollider/sctutorial/5.4%20Granular%20Bonus%20Examples.html