Project report

Arduino piloting for Cardinal Modular Synthesizer

Group 11 :

Thibaut HABERER
Yann MLEKO
Gwénolé MOISON

2023/2024
INFO 4 - Polytech Grenoble

Contents

Introduction

In our 4th year as Information Technology engineering students in Polytech Grenoble, France, we had to manage a project in a small size team. We chose a subject that aimed at creating a physical support using an Arduino Uno board for a virtual modular synthesizer plugin called Cardinal.

The idea behind this project was to be able to pilot in real time the Cardinal synthesizer from the Arduino board using components able to send data and that can be connected to the board.

The solution delivered at the end of this project can be compared in a very simple way to a DJ set linked to Cardinal, allowing to create music or anything related to sound creation while being really inexpensive, unlike classical DJ sets that can be bought.

1. The project

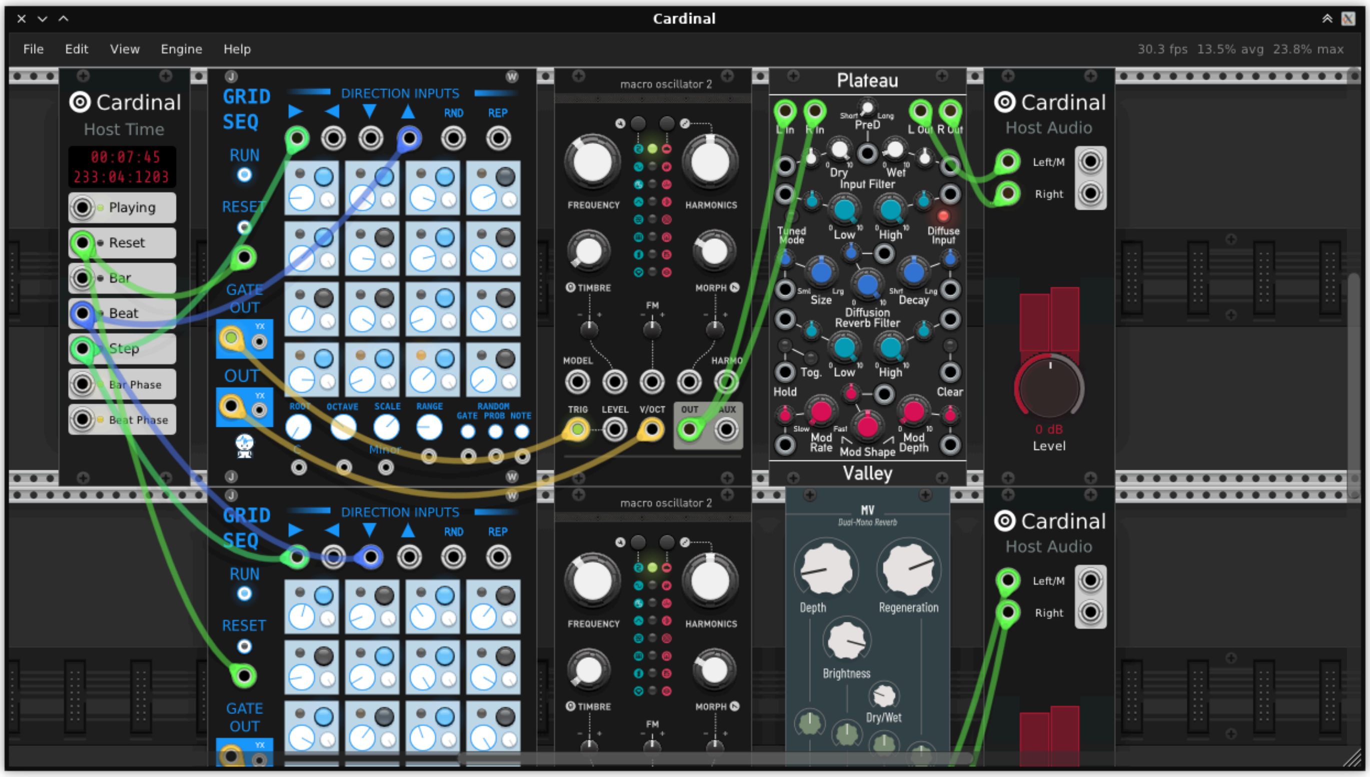
1.1. Presentation of Cardinal Virtual Modular Synthesizer

“A fully free and self-contained modular synthesizer
based on the popular VCV Rack”

Source : Cardinal website main page

Cardinal is an open-source synthesizer based on the proprietary virtual modular synthesizer VCV Rack. It offers a lot of features, including a large list of open-source modules based on VCV Rack official modules.

This synthesizer is implemented in C++ and offers the possibility for developers to implement new features for the synthesizer and even propose to add these features as



Cardinal official features.

1.2. Project description and specifications

As said in the beginning of the report, the project aims at implementing a feature that would allow the communication between an Arduino board and Cardinal. Concretely, an interface must be built to retrieve that data of the components connected to the Arduino board, process this data to a format that can be sent to Cardinal and eventually send the data to Cardinal to make it directly interpretable by the synthesizer. All this in real time so there is a need to ensure that the data is actualized permanently and no delay must appear between the usage of the components of the Arduino board and the transmission of the MIDI message in Cardinal.

A specification of the project is that the solution must do some tasks automatically, such as handling disconnections from the Arduino board or Cardinal crashes. Another issue is that the components of the Arduino board can be disconnected and reconnected during the functioning of the interface. This means handling a wire management between the components of the Arduino card and the modules they are connected to in Cardinal.

This also involves the issue of short-circuits, because the user can miswire the components and create a short-circuit in the Arduino board, leading to its destruction. A solution must be found to take into account this scenario, accepting that we cannot directly control the wiring of the components on the Arduino board.

1.3. Technologies used

Cardinal is implemented in C++, therefore to update the wiring of the modules in function of the wiring on the Arduino board, the code of Cardinal must be modified.

The protocol chosen to convert the raw data coming from the components of the Arduino board is the MIDI protocol, because Cardinal has modules accepting MIDI messages, it has been decided to use this protocol. The raw data received will be transformed into a MIDI message and sent to Cardinal.

To handle the transfer of data from the Arduino board to Cardinal, an interface has been created using Python language and several libraries such as MIDO (MIDI Objects for Python), which allows to create MIDI messages from Python, Serial to retrieve the raw data from the Serial port on which the Arduino board sends its data, and Jack that allows to create and manage audio and MIDI communications channels with ports, based on sockets.

Cardinal itself will also need to be modified in order to update the connections between the modules in the application when changes occur and are sent to the synthesizer from the interface.

1. Arduino setup and wiring

1.1 Arduino Setup and Wiring

To set up your Arduino, wire the four potentiometers to the analog ports of the Arduino board. These ports will be utilized for reading values by the program. The digital ports will serve as input/output (I/O) channels, mimicking the jack connectors of Cardinal. Follow Arduino's standard usage of the digital ports for this setup.

1.2 Outputs

When readings are taken, they will be printed in a specific format. The analog port readings will each be displayed on a new line with a single value. They will be printed sequentially from 1 to 4 (1 corresponding to the potentiometer connected to the first analog port and so on). Digital port readings will follow a different format. They will be displayed as "/x:y" where 'x' represents a value and 'y' is greater than 'x'. The interpretation that you must have of this is that the port x is connected to port y. Note that the last digital pin is reserved for specific purposes.

1.3 Error Detection

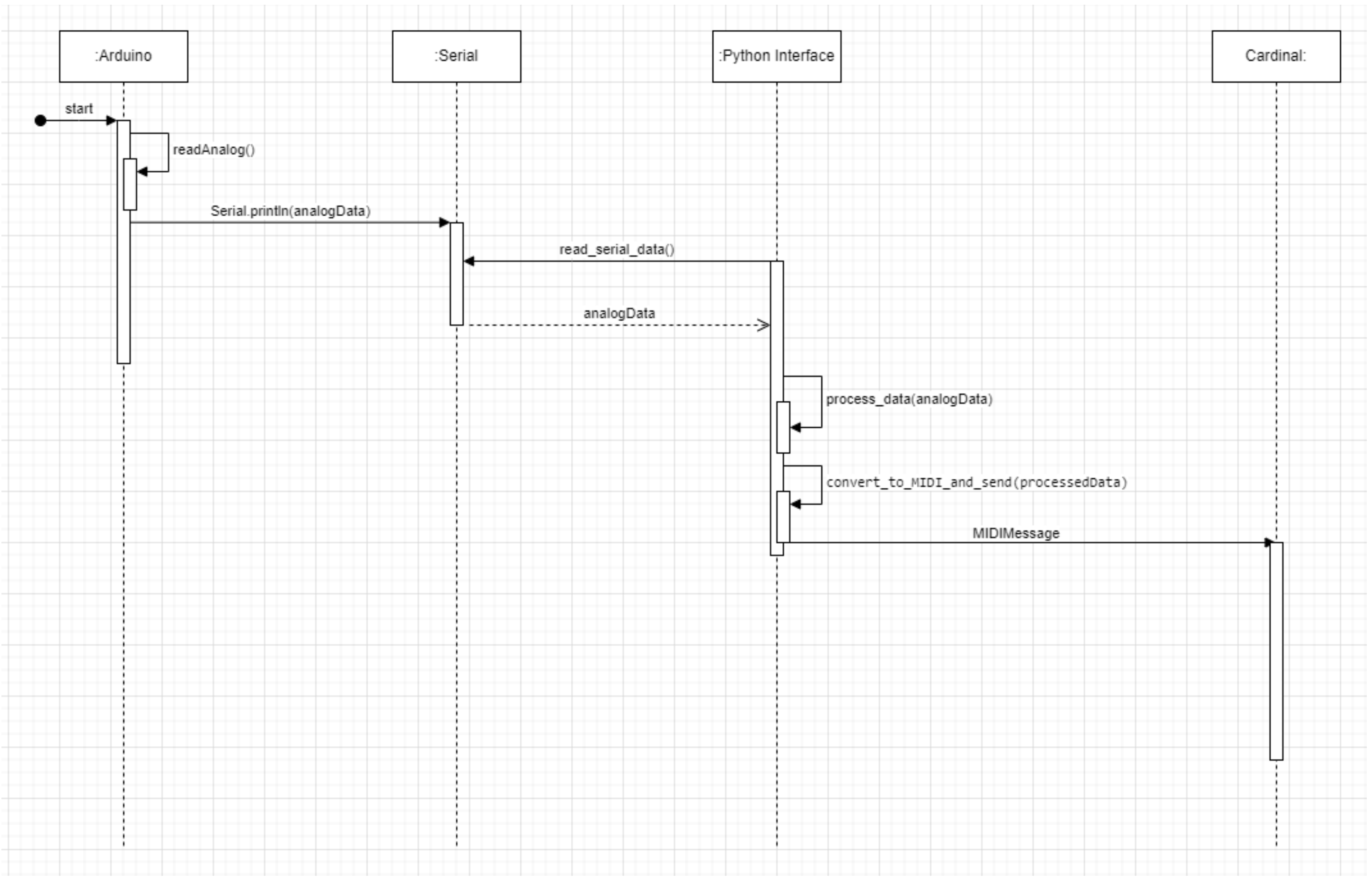
In cases where you need to modelize 9 ports, you'll require 10 digital ports. Arduino may occasionally produce incorrect readings. However, these errors follow a consistent pattern.It will say that everything is connected, thus the last pin should never be connected to other pins. By maintaining this configuration, it allows for the identification of erroneous readings. When the ID of the last pin is observed in the output, it signifies that the reading is incorrect and should not be used.

2. Python interface for MIDI and wiring management

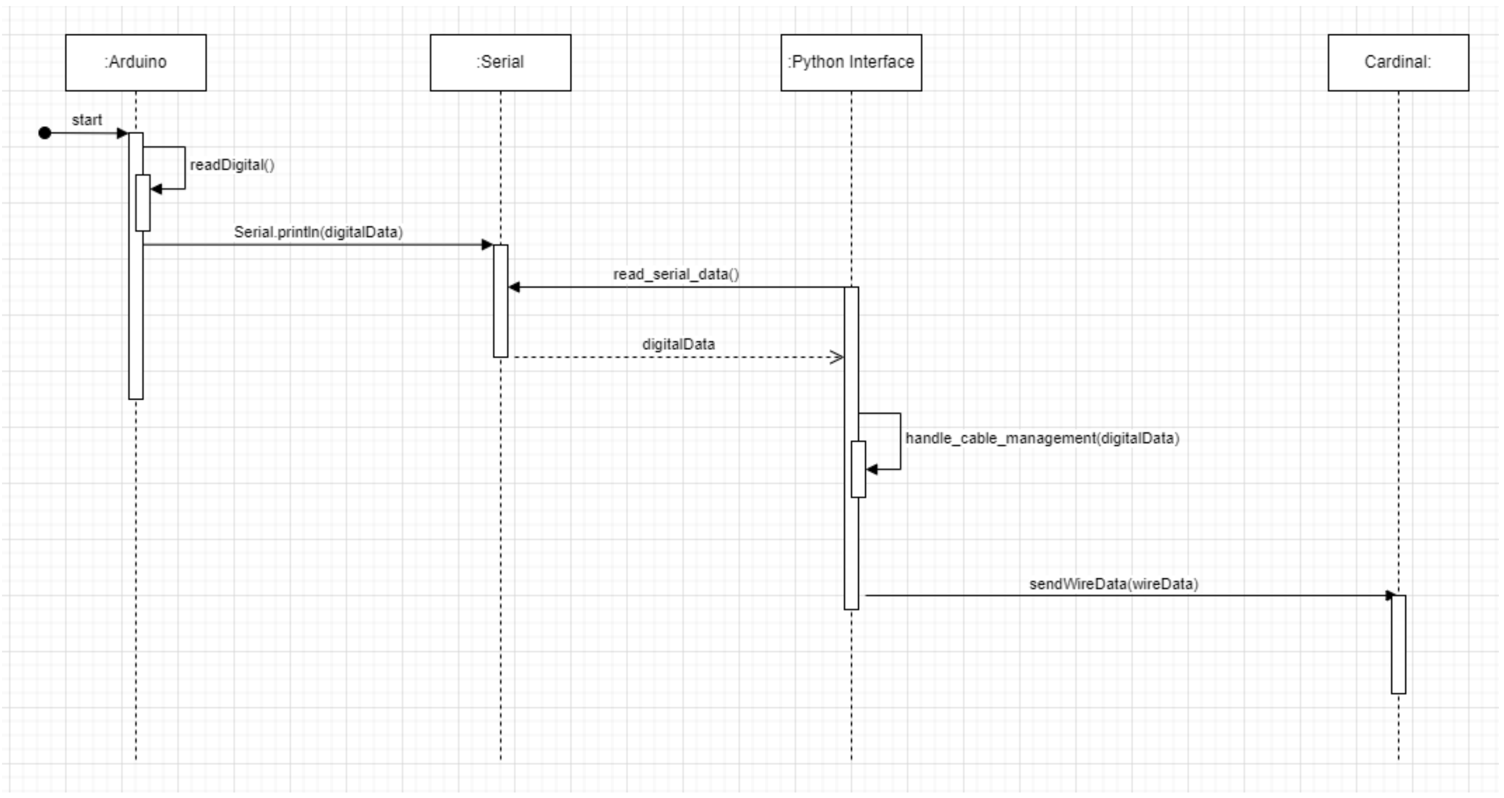
2.1. Sequence diagrams and functioning of the interface

Here are two sequence diagrams describing the two main use cases :

- send MIDI data



- manage wiring



2.2. MIDI processing Implementation

When started, the Python interface opens a new virtual MIDI output port and connects it to the Cardinal's MIDI input port so MIDI messages can be passed through this communication channel.

Audio ports are also set up to display the audio in the system output by default. Doing this allows the audio outputs to be changed, and each channel (for the left and right outputs) can be managed separately.

The implementation of the Python interface allows it to read data from the serial port on which the Arduino board sends its data. This raw data is therefore filtered because Arduino boards tend to send random data at some points, making errors and possibly crashing the system if they are not filtered. After this step, the filtered data is processed in order to be transformed into MIDI messages.

The processing of the data includes two steps :

- The first step is resumed into formatting the received data which is generally coded in 10 bits (values are in range from 0 to 1023). Indeed, Arduino boards components usually use 10 bits data, however Cardinal MIDI ports only received data coded in 7 bits, therefore a small conversion must be done to make the data valid to be sent.
- The second step was added to fix an issue discovered during the tests with the Arduino board. The potentiometers used to generate the raw data were very unbalanced, a range of 50 different values (among the 1024 possible values) were displayed on approximately 75% of the rotation angle of the

potentiometers. That is to say the data was barely changing until the potentiometers were in the last 25% percent of their rotation angle. This issue made necessary the application of a scale factor to increase the change factor of the values. A simple conversion was made so when the rotation angle was in the 75% where the data was barely changing, a scale factor was applied on the data received to increase the values and make a true difference between them. Though it led to a diminution of the values that can be displayed because only a small part of the values are displayed in this range of the potentiometers' rotation range.

After the data has been processed, it is converted into a MIDI message and sent to Cardinal through the MIDI port created by the interface. The MIDI messages are sent to a designated MIDI CC (Control Change) cell in Cardinal. These are basically outputs for MIDI messages in Cardinal which can be used from external devices in order to communicate with Cardinal. Each CC cell corresponds to a component connected to the Arduino board.

Eventually the MIDI messages are received in Cardinal and can be used freely in the synthesizer, by connecting the CC cell where the MIDI message is output to any available module.

The interface also handles crashes from both Cardinal and the Arduino card, trying to reconnect to them until a timer is reached. This allows the interface to be more reliable, robust, and fault resistant if the user unwillingly closes Cardinal or disconnects the Arduino board.

2.3. Wiring autoconfiguration implementation

The interface needs to transmit the wiring data to Cardinal in order for it to change the cable position. To do so, we need to filter the data in the correct format.
Each Pin has been given an ID and correspond either to a module input or a module output:

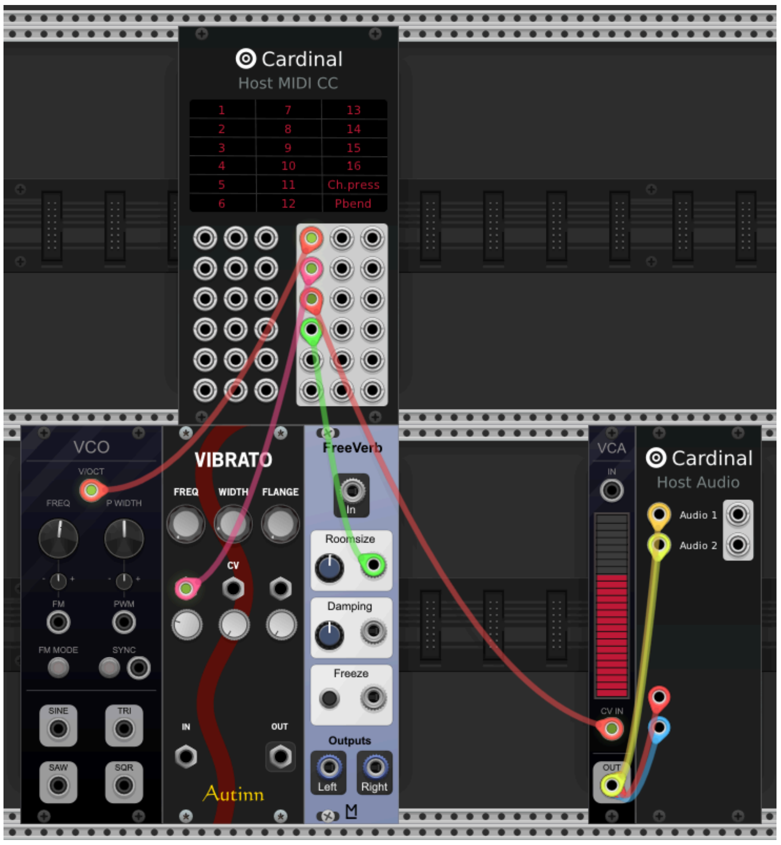
1. VCO SIN
2. VCO TRI
3. VCO SAW
4. VCO SQR
5. VIBRATO IN
6. VIBRATO OUT
7. FreeVerb IN
8. FreeVerb Output Left
9. VCA IN

We filter the data sent by the Arduino to avoid errors. If we obtain an ID not corresponding to this list, the wire update is not sent to Cardinal. Because you can't connect two module outputs or two input modules together, we filter if this is the case. We also format each cable data to have the module output first and then the module input.
If there is no error, the socket sends the wiring data to Cardinal.

3. Modifying Cardinal

3.1. The Cardinal setup

Modifying the source code of Cardinal is a necessity in our case because the control of cables can't be handled using the MIDI Protocol like the other parameters. To be able to handle the cables correctly, we first need to define the default configuration.

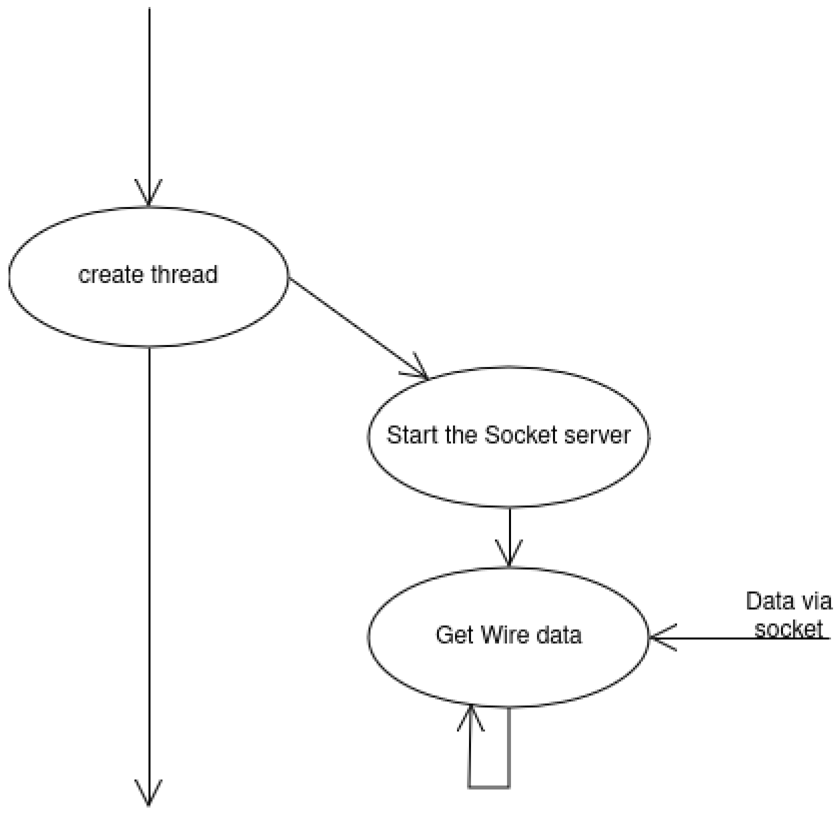


We created a custom configuration with a module generating different types of waves, a module adding a vibrato effect and one adding reverb to the signal. We combine these three modules with a MIDI Control Change module, a module that allows us to control the output volume and Computer Audio output. All the cables that are shown on the above picture are either used to transmit the potentiometer value to the corresponding pin or to output the audio. These cables can't be removed by the interface.

The Class CardinalCommon.cpp initializing the application has been changed to load the save file corresponding to this configuration instead of the default new file.

3.2. Communicating with the interface

At launch of the application, the modified version of Cardinal starts a socket server that is used to catch the wiring data sent by the interface. When the socket connection is accepted, a thread is created to catch the data without blocking the whole application. At reception each cable that isn't in custom configuration is removed, then the cables are set according to the data received.



4. Suggestions of improvements

The solution is almost functional according to the base specifications of the project, but there are several points that can be improved upon. We didn't manage to truly implement the connection between the interface and Cardinal. Uploading Cardinal to the repository resulted in a loss of files due to lots of ignored files in the source code. We were also unable to launch the socket and jack's socket at the same time due to a connection problem.

The configuration of MIDI CC cells to which the components of the Arduino card are linked is manual, an improvement could be to implement an auto configuration at launch, based on a configuration file, or allow the user to easily define which cell corresponds to which component.

Another point would be to increase the number of components that can be connected to the Arduino board, because currently the number of wires managed by the interface is relatively small, therefore only a few components can be connected and handled.