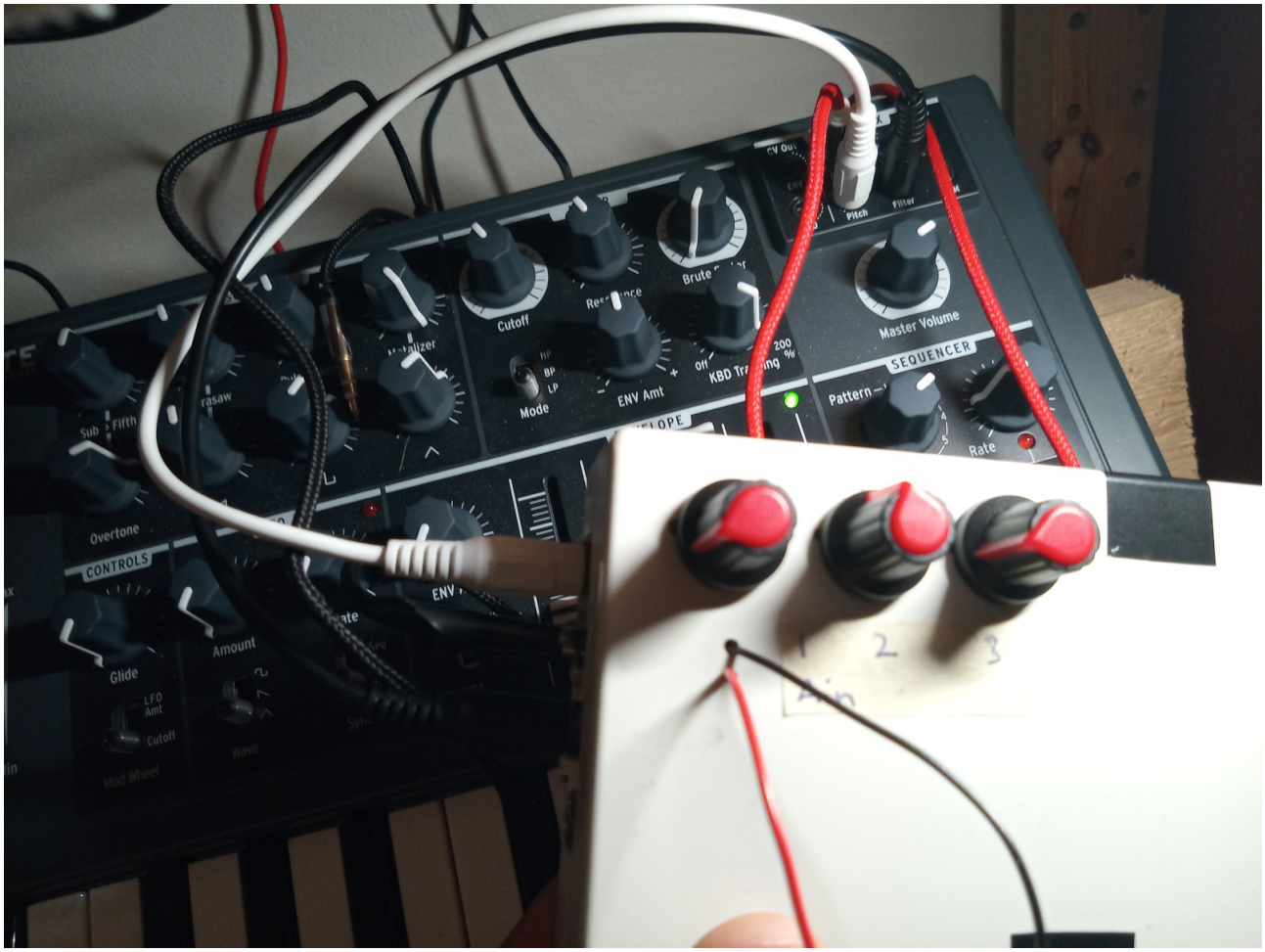
OscCVGateMod

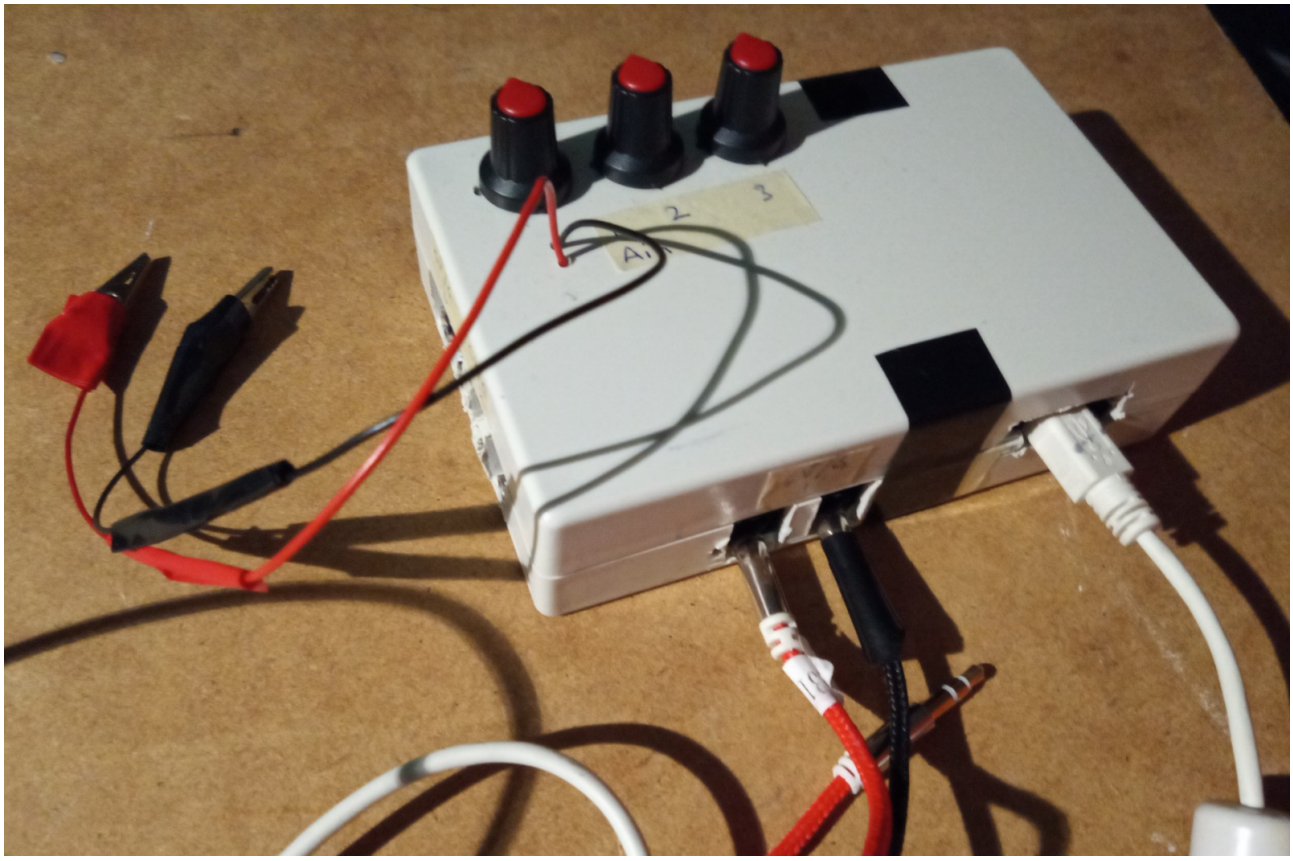
3+3-channel CV/Gate modulator

Project: OscCVGateMod
Author: Staffan Melin, staffan.melin@oscillator.se
License: GNU General Public License v3.0
Version: 20200313
Project site: http://www.oscillator.se/arduino



Contents

Introduction



The OscCVGateMod is a 3+3 channel digital/analog CV/Gate modulator based on an Arduino.

An experimental platform for manipulation of CV/Gate synthesizers.

It has 3 independent CV channels that operate from 0 to +5V to drive up to 3 analog synthesizers. It also has 3 independent gate (clock) channels outputting 0 or +5V. It can be thought of as a kind of programmable LFO.

The signals are controlled by software, using algorithms contained in different classes. These classes can easily be expanded and changed, adding additional capabilities.

The algorithms can also be controlled using:

- one or more of 3 potentiometers
- an analog input

The analog input is by default used as a clock input, making it possible to synchronize the algorithms to external devices.

The lack of display and editing capabilities means that major changes and configuration has to be done in the Arduino IDE. To use the OscCVGateMod you have to have some knowledge in programming the Arduino.

How to use it

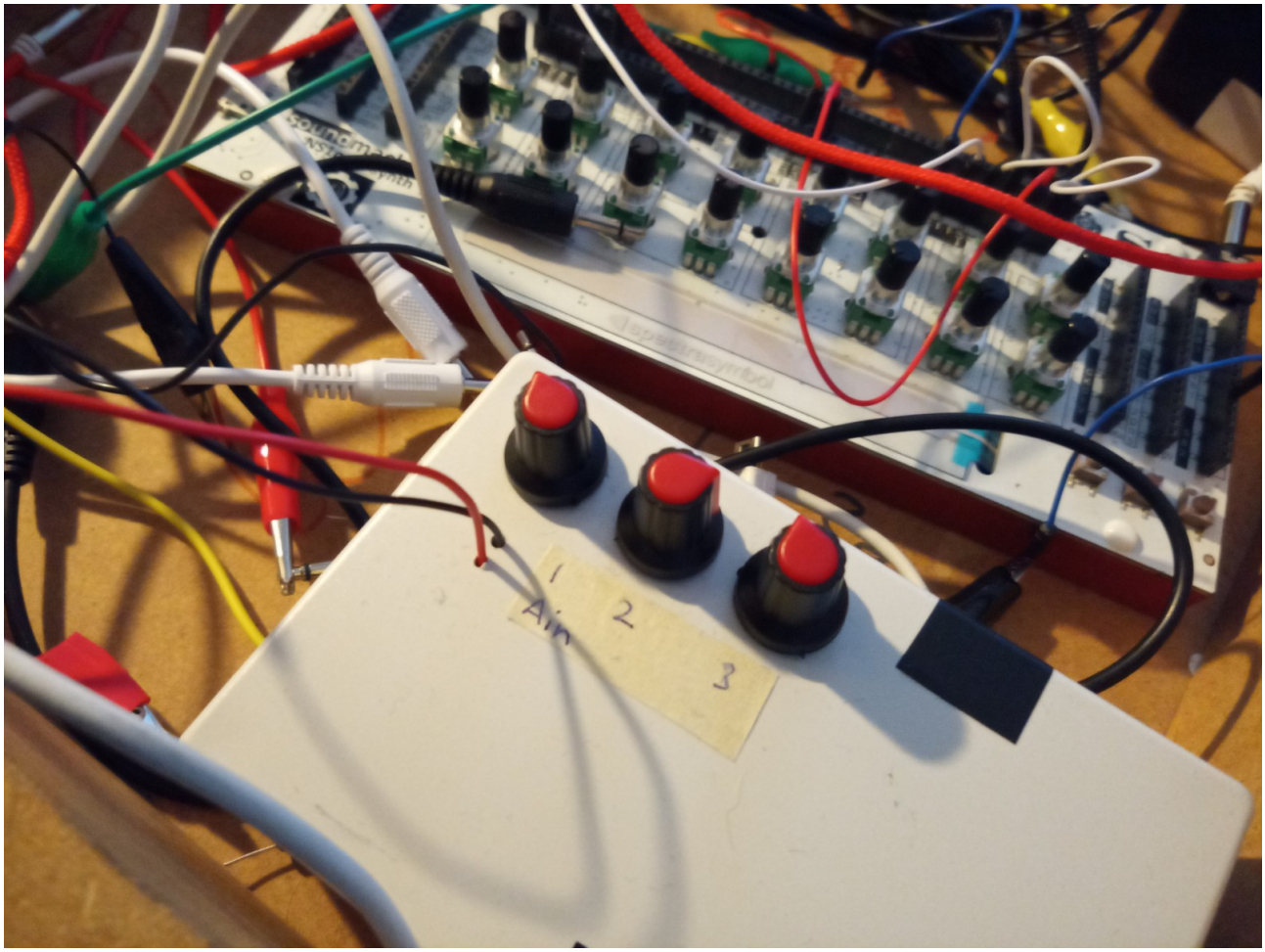
To use the OscCVGateMod you must have some basic skills in Arduino and the Arduino IDE.

All modulation takes place in classes. There is one class for each type of modulation.

Every class has (at least) four methods:

- init() - initializes the modulation
- work() - called in the Arduino loop() to calculate the signal value
- getValue() / getGate() - get the current value of the modulation
- setValue() / setGate() - set the value of the modulation

In several classes the actual computation of the (new) modulation value is done in a calcValue() function.



Modulating CV

The 3 CV modulating objects are created before the setup():

```
CVWanderer cv1;

CVTriangle cv2;

CVWanderer cv3;
```

They are initialized in setup():

```
cv1.init(false, 0, MAX_DAC, 1, 10, PIN_POT_1, PIN_POT_NONE); // CVWanderer

cv2.init(false, 0, MAX_DAC, 2000, PIN_POT_2); // CVTriangle

cv3.init(false, 0, MAX_DAC, 1, 10, PIN_POT_NONE, PIN_POT_NONE); // CVWanderer
```

where you can find examples on initialization of the different modulation classes.

The arguments are described at each class definition.

Modulating Gate

The 3 gate objects are created before the setup():

```
GateFixed gate1;
GateClock gate2;
GateRandom gate3;
```

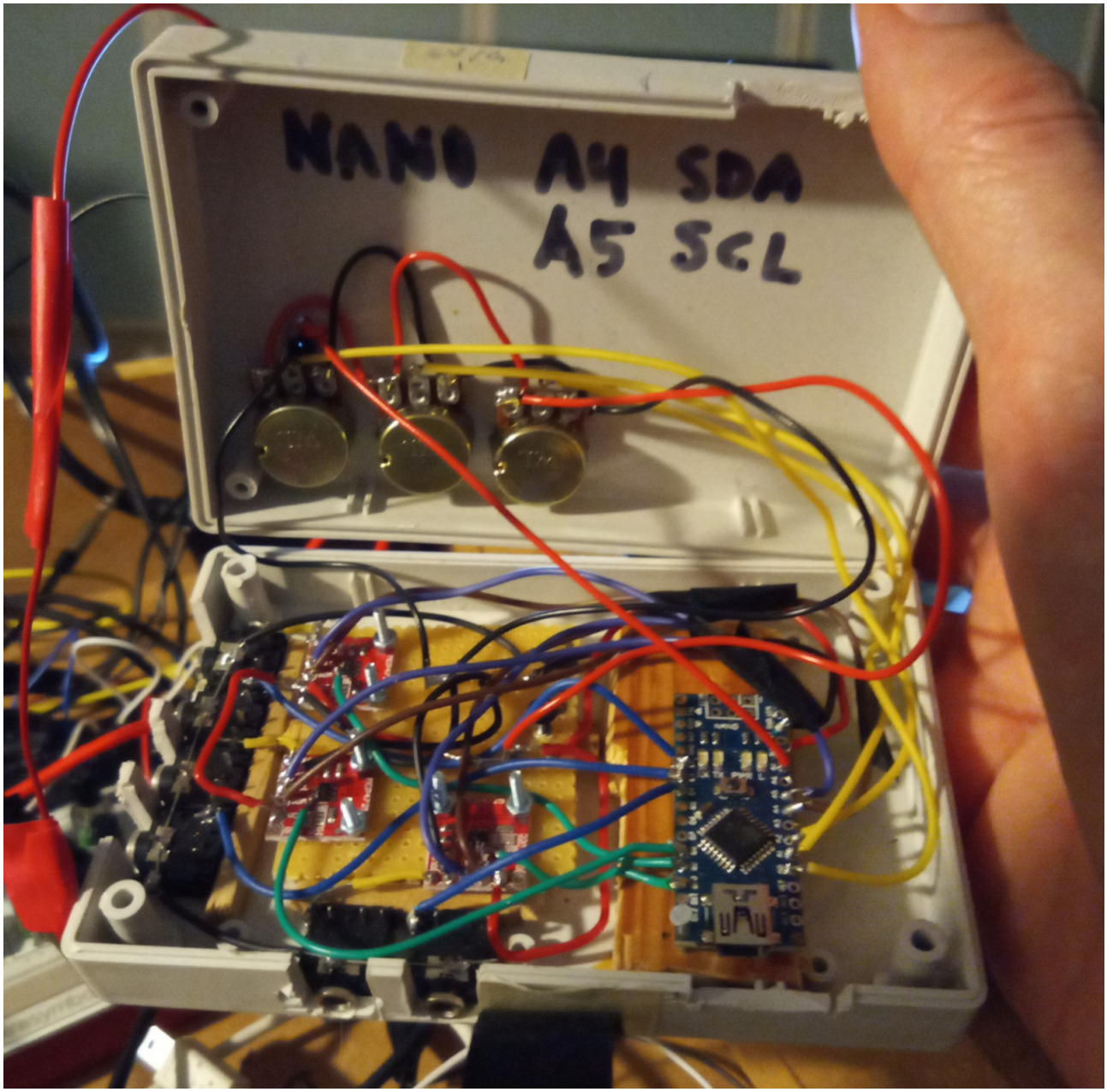
They are initialized in setup():

```
gate1.init(false, true); // GateFixed
gate2.init(false, t16, 50, PIN_POT_NONE); // GateClock
gate3.init(false, t16*2, 50, PIN_POT_3); // GateRandom
```

where you can find examples on initialization of the different gate classes.

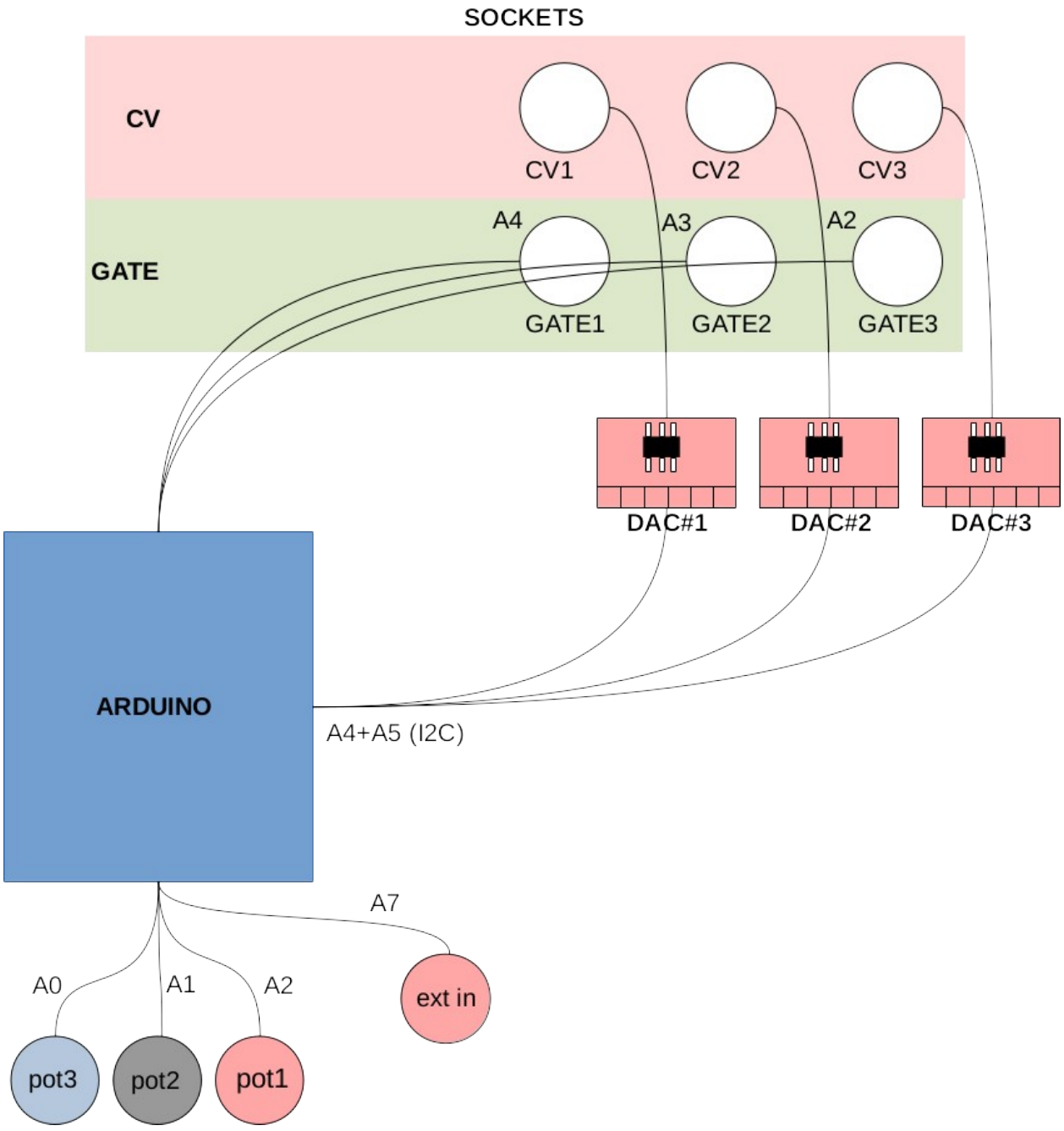
The arguments are described at each class definition.

How to build it



The are a lot of connections to be made, so I recommend the following order, testing each component before moving on to the next.

System overview:

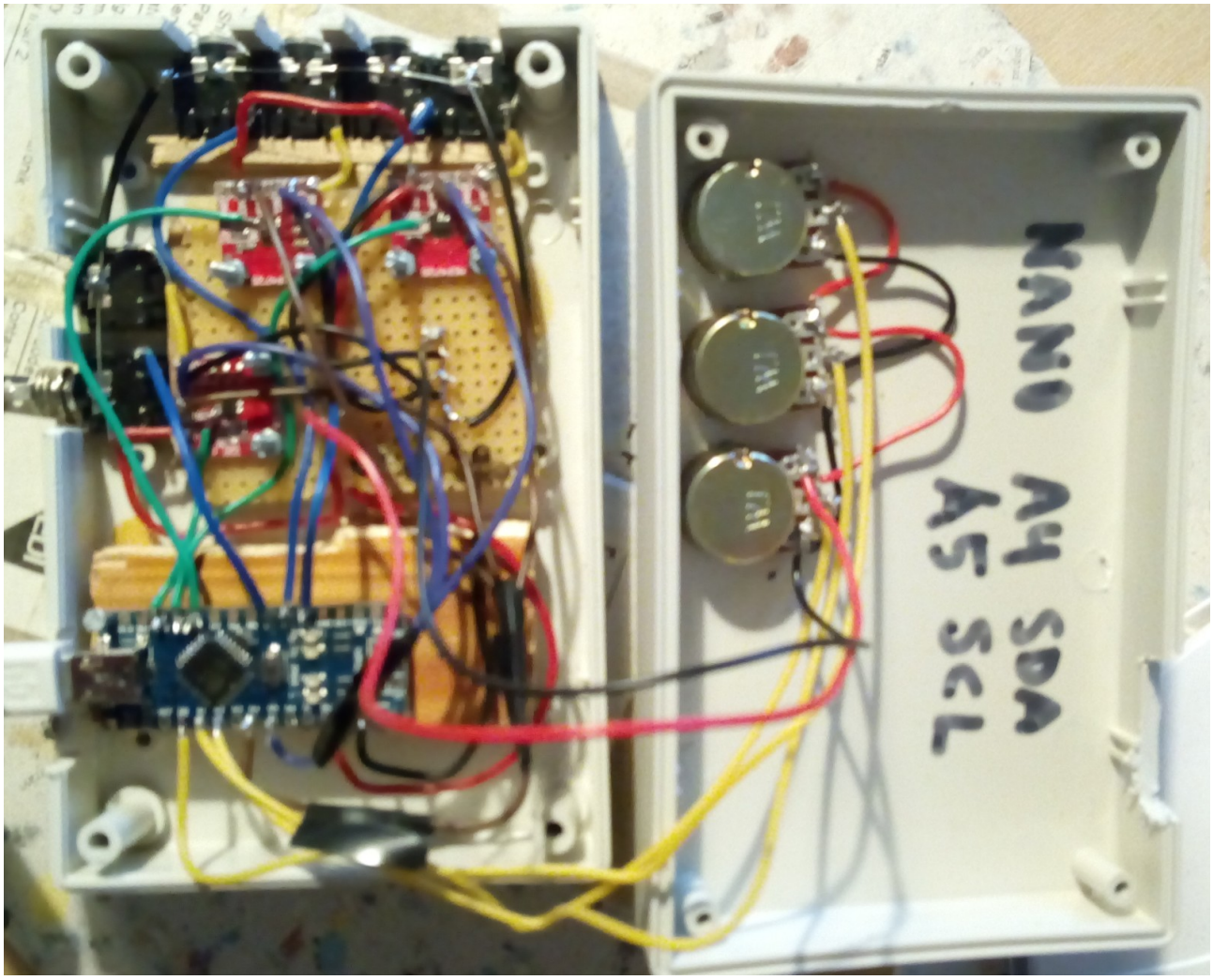


Equipment

- 1 x Arduino (I used an Arduino Nano, but an Uno R3 will also work)
- 3 x DAC MCP4725
- 6 x 3.5mm mono audio sockets (female)
- 3 x potentiometers (I used 10k Ohm)
- prototyping board/perf board
- connection wires
- enclosure

You also need the usual tools such as a soldering iron/station, multimeter etc. And at least one analog synthesizer with CV/Gate connections.

If you use an Arduino Uno R3, you have to replace the connection to A7 (external gate in) with A3, in both circuit and code.



Potentiometers

The three potentiometers are connected to GND and +5V on the Arduino. The middle pin is connected to

- Pot 1 - Arduino A2
- Pot 2 - Arduino A1
- Pot 3 - Arduino A0

DACs

The OscCVGateMod is made to control analog synths based on the 1V/octave scheme. As the DAC can output voltages from 0 up to what it is fed, in our case the +5V from the Arduino, we can control 5 octaves. This means 5 octaves x 12 notes = 60 notes. If you want to transpose these 5 octaves you have to do it using the oscillator tuning on your synthesizer.

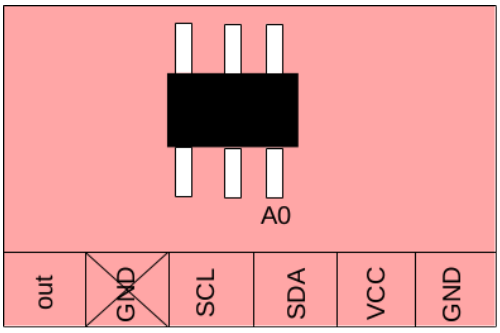
The Arduino controls the Digital to Analog Converters (DACs) using the I2C protocol. For this to work the DAC boards must have unique addresses.

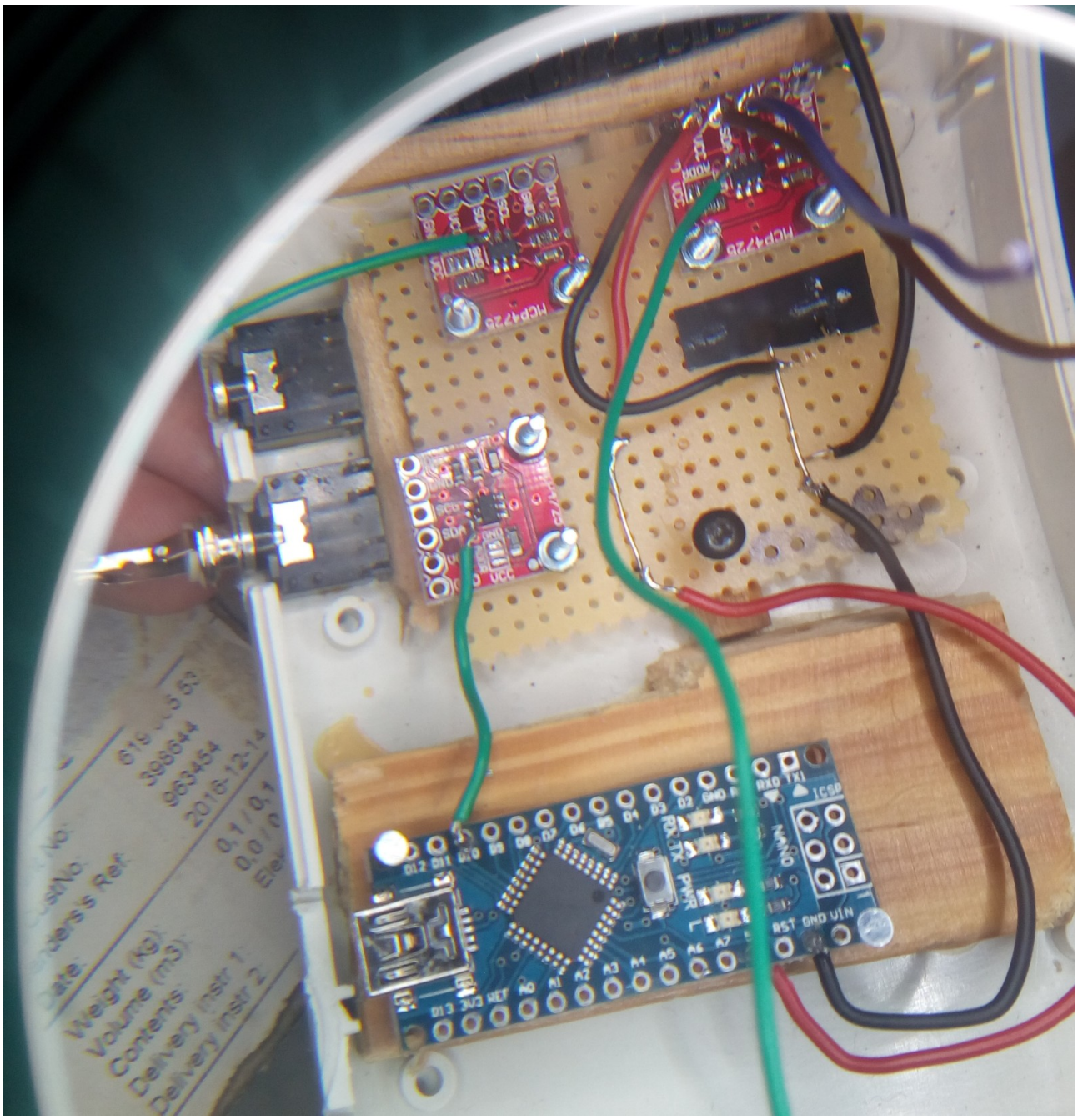
I used the common MCP4725 DAC. When you buy these they are usually set to a specific address, with the option to increase the address by one by setting the A0 pin on the DAC board to HIGH. *Do not confuse this with the A0 pin of the Arduino.*

You can find out which I2C address your DACs use by running the program i2c_scanner.ino (search for it on the web).

In my case the two addresses that can be used are 0x62 and 0x63. As we want to control 3 channels we do this by setting A0 to LOW on the DAC we want to use and HIGH on the rest. We then only write to the 0x62 address. This is managed by the dacSelect() function in the code.

The A0 pin was not exposed on my DAC board to I had to solder a wire directly to the A0 leg of the DAC chip.




MCP4725
MCP4725
MCP4725
Weight (Kg)
Volume (m3)
Contents
Delivery instr 1
Delivery instr 2
2016-12-14
RST GND VIN

Connect the DACs:

DAC pin	Use	Connects to
DAC #1		
Out	output	CV 1 (tip)
GND	not used	
SCL	I2C communication	Arduino A5
SDA	I2C communication	Arduino A4
VCC	+5V	Arduino +5V
GND	GND	Arduino GND
DAC #2		
Out	output	CV 2 (tip)
GND	not used	
SCL	I2C communication	Arduino A5
SDA	I2C communication	Arduino A4
VCC	+5V	Arduino +5V
GND	GND	Arduino GND
DAC #3		
Out	output	CV 3 (tip)
GND	not used	
SCL	I2C communication	Arduino A5
SDA	I2C communication	Arduino A4
VCC	+5V	Arduino +5V
GND	GND	Arduino GND

Test each DAC connection using the test code: code_test_dac.ino.

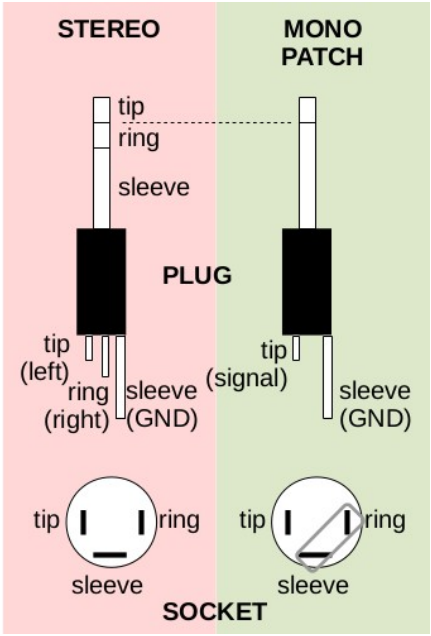
I have read that you need to disable the pullup resistors on all but one DAC if you use more than one, but I didn't do this (https://learn.sparkfun.com/tutorials/mcp4725-digital-to-analog-converter-hookup-guide).

Links

- https://www.instructables.com/id/ADAFRUIT-MCP4725-FOUR-CHANNEL-SETUP/

Sockets

You need 6 x 3.5mm female mono sockets. I used stereo sockets and shorted the ring and the sleeve.



Connect the sockets:

- sleeve (+ring) - GND
- CV1 tip - DAC#1 out
- CV2 tip - DAC#2 out
- CV3 tip - DAC#3 out
- Gate 1 tip - D4
- Gate 2 tip - D3
- Gate 3 tip - D2

External clock in

I decided to connect the analog in line to a crocodile clip for freedom of connection. In the future I might want to try to connect an analog sensor.

You can also use an 3.5mm socket.

I also decided to connect ground to a crocodile clip. In this way the two clips can connect to an male audio/CV cable.

Arduino connections

Arduino pin	Use	Connects to
D2	Gate #3	Gate #3 socket tip
D3	Gate #2	Gate #2 socket tip
D4	Gate #1	Gate #1 socket tip
D8	CV #3 DAC select	DAC #3 A0 pin
D9	CV #2 DAC select	DAC #2 A0 pin
D10	CV #1 DAC select	DAC #1 A0 pin
A0	Pot 3	Pot 3 middle pin
A1	Pot 2	Pot 2 middle pin
A2	Pot 1	Pot 1 middle pin
A3	*use as Ext clock in on Arduino Uno*	
A4	DAC SDA	SDA port of all 4 DACs
A5	DAC SCL	SCL port of all 4 DACs
A7	Ext clock in	Crocodile clip input

Code

The software makes use of libraries that you can install in the Arduino IDE:

```
#include <Wire.h>
#include <Adafruit_MCP4725.h>
```

If you want or need to know more about object oriented programming and the Arduino, this is a good resource: http://paulmurraycbr.github.io/ArduinoTheOOWay.html.

Ideas for improvement

- Add a display and input buttons so you can configure which algorithms work with the different connections.
- Change the DACs so they can handle more than +5V, making the OscDigiSeq handle more octaves.
- Use this as a hardware platform for exploring other ways of controlling analog synths: drone firmware, art projects etc.