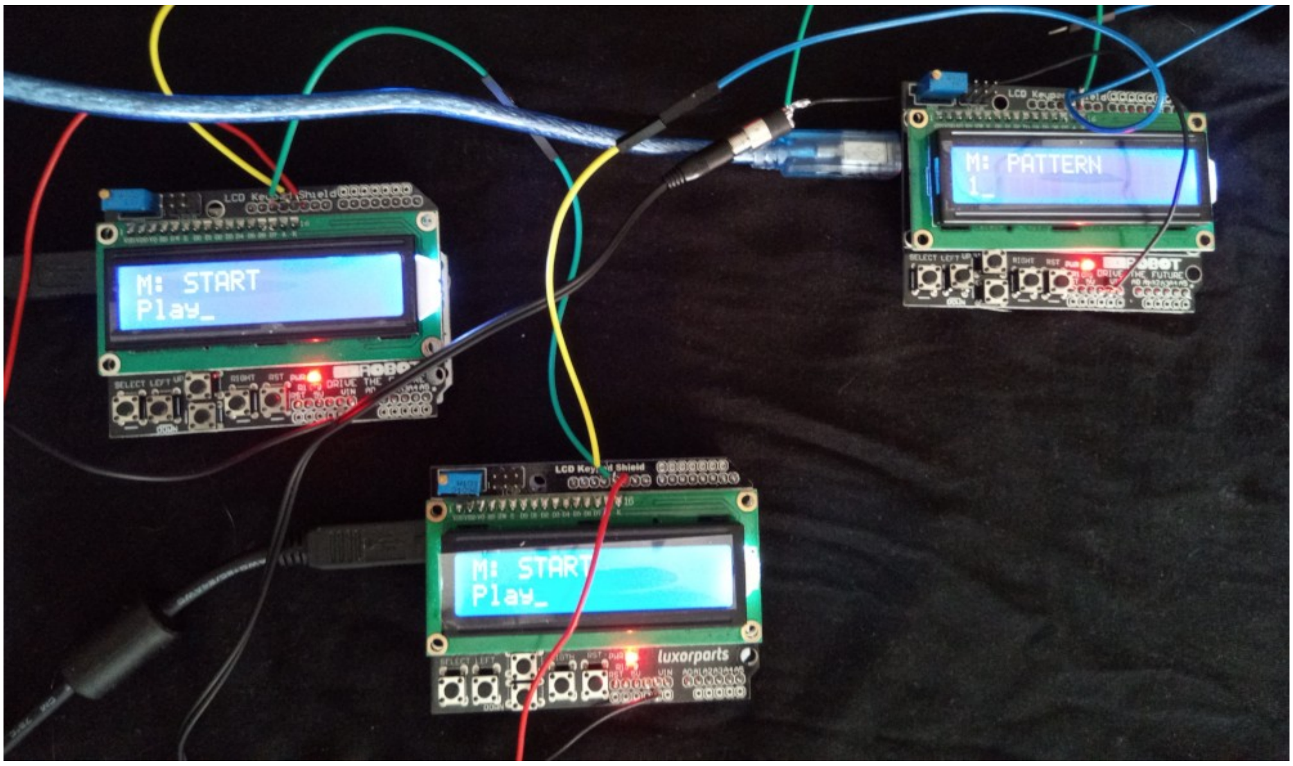
OSCPOCKETO

Version: 2023-12-13



FRIPROGRAMVARU-
syndikatet

Introduction

Welcome to the OscPocketO - Arduino Pocket Synth!

The OsckPocketO (OPO) is a family of affordable and portable sound generators running open source software!

All software is running on the Arduino microcontroller, including sound generation thanks to the awesome Mozzi library[1]!

This guide assumes you know how to connect, edit and send sketches to an Arduino. If not, check out the documentation[2].

It also assumes that you have installed the Arduino IDE (see “Install the Arduino Desktop IDE”) and the following libraries[3]:

- Liquid Crystal
- Mozzi

OPO is currently two different machines: OPO Synth and OPO Drums. They both use the same hardware, so if you build this machine you can change how it operates by uploading either of sketches!

You can download this instruction and all the code you need from <https://oscillator.se/arduino/#oscpocketo>.

You can find a lot of great resources for learning to work with the Arduino in their documentation[4].

Synth features

The OPO Synth features:

- a 16 step sequencer with multiple patterns
- adjustable tempo and gate length
- four selectable waveforms
- settings for attack and decay
- a low pass filter with modulation and cutoff and resonance settings

1 https://sensorium.github.io/Mozzi/

2 https://support.arduino.cc/hc/en-us/articles/4733418441116-Upload-a-sketch-in-Arduino-IDE

3 https://docs.arduino.cc/software/ide-v1/tutorials/installing-libraries

4 https://docs.arduino.cc/learn

- an optional second detunable oscillator
- the ability to save and load synth settings and patterns to EEPROM (a memory space that does not disappear when the Arduino is powered off)
- functions to create patterns
- a play mode for solo play

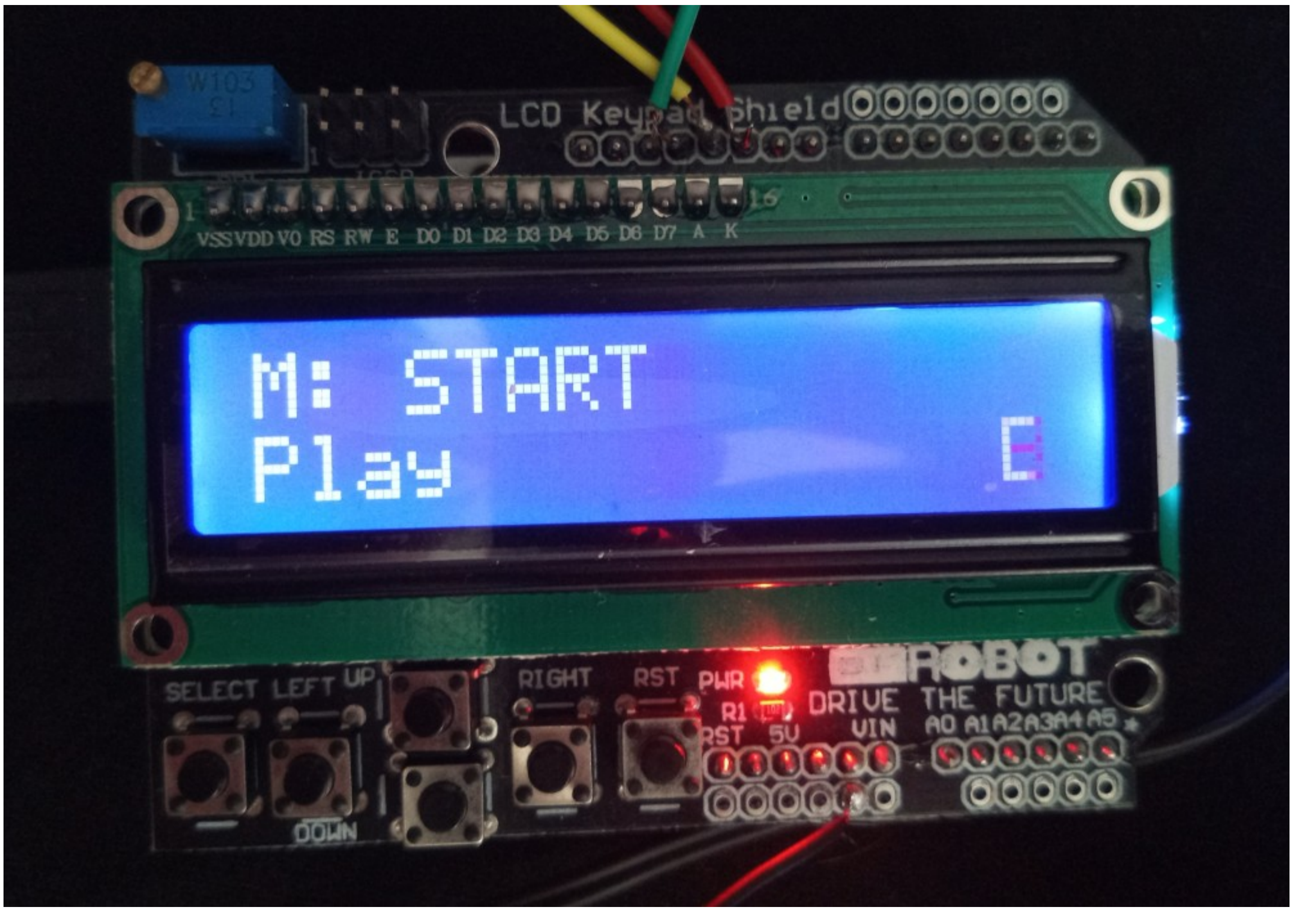
Drums features

The OPO Drums features:

- virtual analog generated sounds: Kick, Snare, Hihat, Clap, Crash, Tom
- a 16 step sequencer
- multiple patterns
- adjustable tempo
- many settings for tweaking the drum sounds
- the ability to save and load patterns to EEPROM
- functions to create patterns
- a play mode for solo play

Table of Contents

How to use it



Figur 1: Overview front

The OPO is controlled by switching to different modes using the SELECT button.

Use the UP button to increase a value, DOWN to decrease a value, and LEFT and RIGHT to move the cursor.

Info:The Arduino's built in LED blinks every time the OPO plays a note.

Warning: Beware that connecting the OPO directly to your home stereo might overload it! Use headphones or a mixer.

Info: If the LCD display messes up press repeatedly so you pass the Tools menu - the LCD will be reset.

Synth

The OPO synth can play patterns you enter, and it has the the following modes of operation.

Modes:

START. Starts and stops the sequencer.

SYNC. Sets the sync mode. NONE = no sync signals are received or transmitted. INT = internal, the built in clock of the OPO is used and sync signals are sent (Conductor mode). EXT = external, the OPO sequencer is controlled by an external signal, but sync signals are still sent (Player mode). EXT24 = as EXT, but the OPO is expecting 24ppq (note: works poorly when tempo > 140 bpm).

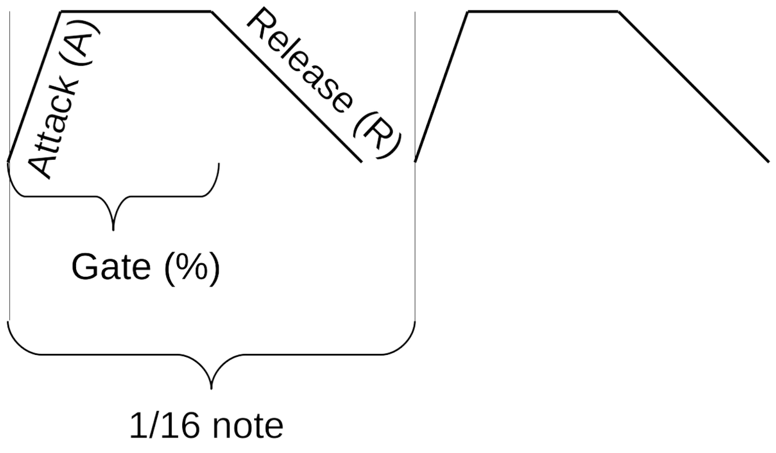
PATTERN. Select the current pattern.

EDIT. Edit the current pattern. Notes are stored as MIDI values in 1 bar (16 x 1/16th notes).

STATE. Edit the state of the notes: X = on, O = off.

TEMPO. Set tempo of the sequencer.

GATE. Set gate of notes played. Gate is expressed as percent of 1/16th.



Figur 2: Illustrating Attack, Gate and Release

SHIFT. Transpose (UP/DOWN) and Shift the sequence (LEFT/RIGHT).

WAVEFORM. Set the waveform of the (first) oscillator: SIN (sine), TRI (triangle), SAW (sawtooth) and SQUARE (square).

ATTACK. Set the Attack time in ms.

RELEASE. Set the Release time in ms.

FILTER MODE. The OPO has a low pass filter. The Cutoff can be modulated:

- FIXED. No modulation, use the Cutoff and Resonance values.
- RANDOM. Random modulation from 0 up to the Cutoff value.
- SLOW. Modulation over approximately 4 bars from 0 to 255. Changes the Cutoff value.

- FAST. Modulation over approximately 1 bar from 0 to 255. Changes the Cutoff value.
- POTS. The cutoff and resonance is controlled by two potentiometers (see section “Expansions” on page 22 in this document).

CUTOFF. Set the Cutoff frequency of the filter (as a number from 0 to 255).

RESONANCE. Set the Resonance of the filter (as a number from 0 to 255).

WAVEFORM2. Activate and set the waveform of the second oscillator: NONE, SIN (sine), TRI (triangle), SAW (sawtooth) and SQUARE (square).

DETUNE2. Detune the second oscillator relative to the first. The value is in Hz and is added to the frequency of the first oscillator.

PLAY. Keyboard mode. The sequencer is stopped (if running) and the 4 first notes of the current pattern are mapped to LEFT, UP, DOWN and RIGHT for solo play.

TOOLS. Small utility functions. Activate with UP.

- S. Save synthesizer settings and patterns to EEPROM so they can be recalled after power off.
- L. Load synthesizer settings and patterns from EEPROM.
- R. Create Random pattern.
- B. Create a Bassline pattern based on the current note.
- C. Copy current pattern to next pattern position.

Drums

The OPO Drums can play 5 simultaneous sounds, all created by virtual analog synths thanks to the Mozzi library: Kick, Snare, Hihat, Clap and Crash.

Modes:

START. Starts and stops the sequencer.

SYNC. Sets the sync mode. NONE = no sync signals are received or transmitted. INT = internal, the built in clock of the OPO is used and sync signals are sent (Conductor mode). EXT = external, the OPO sequencer is controlled by an external signal, but sync signals are still sent (Player mode). EXT24 = as EXT, but the OPO is expecting 24ppq (note: works poorly when tempo > 140 bpm).

PATTERN. Select the current pattern.

EDIT. Edit the current pattern. Notes values are constructed by adding values that corresponds to different sounds:

- Kick = 1

- Snare = 2
- Hihat = 4
- Clap = 8
- Crash = 16
- Tom = 32

An example: A value of 17 means that this step will play Kick (1) and Crash (16), 1 + 16 = 17.

TEMPO. Set tempo of the sequencer.

EDIT KICK. Set frequency of kick, release time and slope (how quickly the sound drops in frequency) where larger value = quicker drop.

EDIT SNARE. Set frequency of snare, release time and slope (how quickly the sound drops in frequency) where larger value = quicker drop.

EDIT HIHAT. Set frequency in some interesting stepped values and release time.

EDIT CLAP. Set release time.

EDIT CRASH. Set release time.

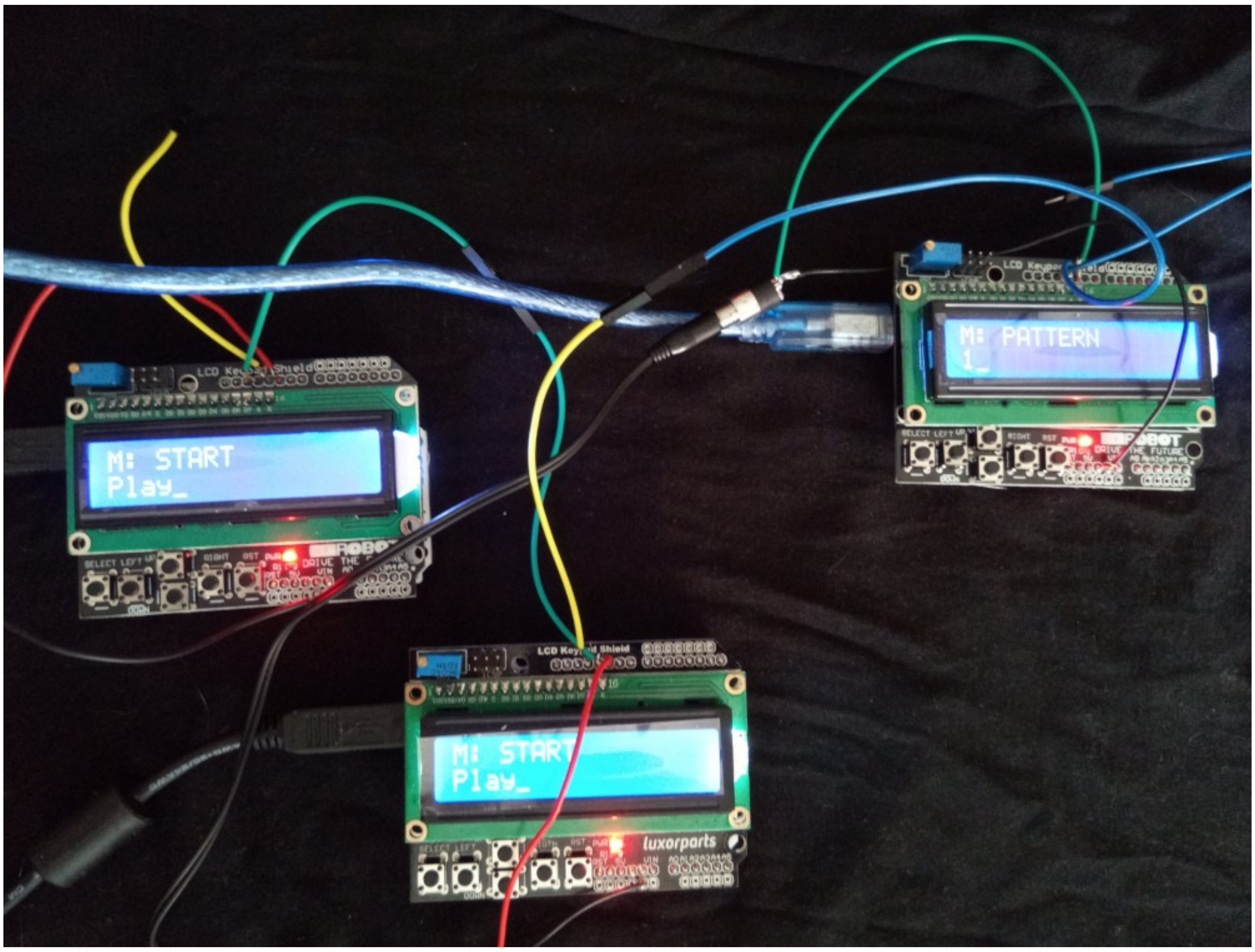
EDIT TOM. Set frequency of tom, release time and slope (how quickly the sound drops in frequency) where larger value = quicker drop.

PLAY. Solo play mode. LEFT = Kick, UP = Snare, DOWN = Tom and RIGHT = Crash.

TOOLS. Small utility functions. Activate with UP.

- S. Save patterns to EEPROM so they can be recalled after power off.
- L. Load synthesizer settings and patterns from EEPROM.
- R. Create Random pattern.
- B. Create a repeating pattern based on the current note.
- C. Copy current pattern to next pattern position.

Syncing several OPO machines



Figur 3: Overview connections for syncing several OPO machines

One OPO has to be the Conductor. This is the machine that sends synchronization data to the other OPOs called Players.

1. Setup

Conductor. Start: Stop. Sync: Internal.

Player(s). Start: Stop. Sync: External. Start: Play. (Order is important.)

Info: Connect SYNC OUT from Conductor to SYNC IN of first Player. Connect GND between Conductor and Player.

If you have several Players connect SYNC OUT from the first Player to SYNC IN on the second Player. Repeat for each Player. Also connect GND between all SYNCed OPO machines.

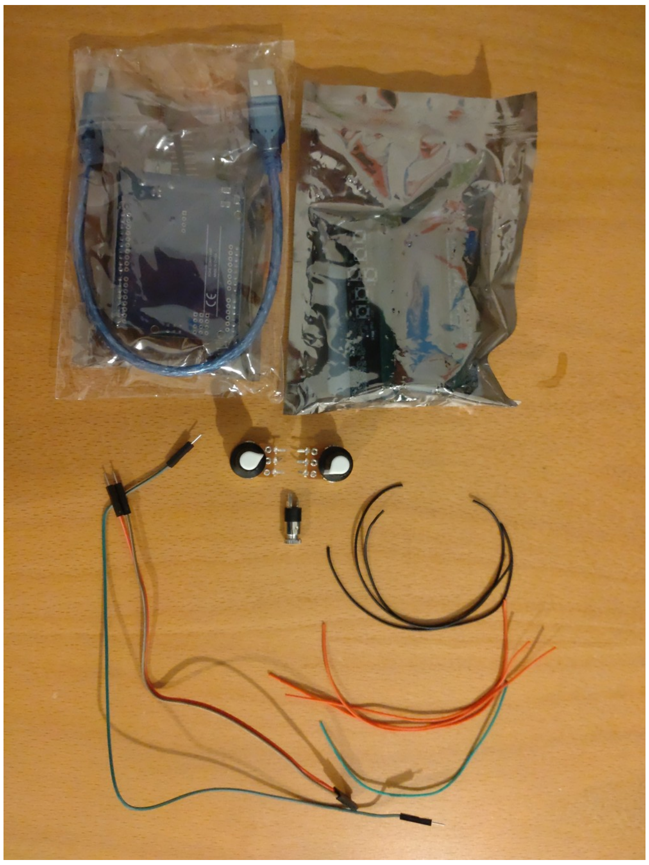
2. Play

Conductor. Start: Play.

You can tweak sounds and switch patterns on all OPO machines. You change tempo (only) on the Conductor.

How to build it

Contents

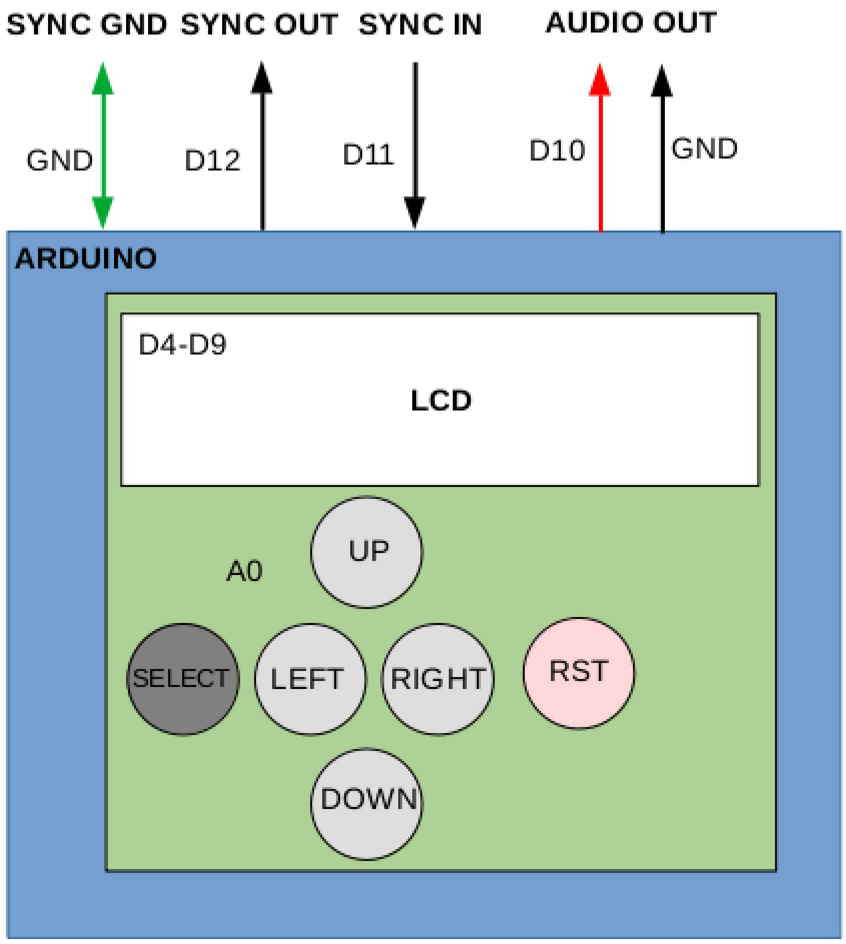

Figur 4: Contents of package

- Arduino Uno R3 (an extra pin list might be included, this is not used in this project)
- a USB cable
- LCD Keypad shield
- Audio/headphone jack (3.5 mm female)
- 2 x potentiometers
- 2 x female – male wires (colors and length can vary)
- 1 x male – male wire (colors and length can vary)
- 1 x red wire 12 cm
- 2 x black wire 12 cm
- 1 x wire in another color (nor red nor black, let us call it “colored”)

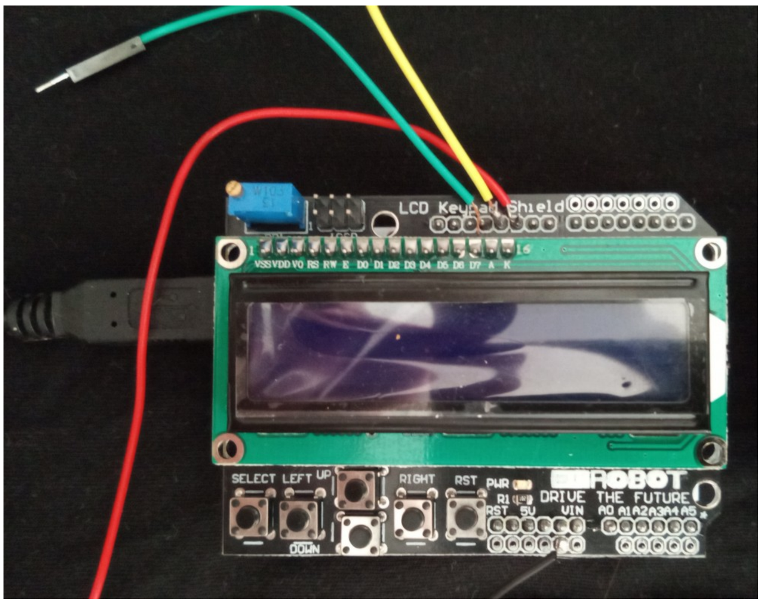
You connect different things to the Arduino by connecting them to “pins” on the Arduino.

Read more about the features of the Arduino here: <https://docs.arduino.cc/tutorials/uno-rev3/intro-to-board>

Info: Overview of connection *(the result after following the steps in this manual, don’t worry if you at this stage don’t understand everything)*:



Figur 5: Schematic of connections and buttons



Figur 6: Cables connected to pins according to schematic (not SYNC GND)

Equipment

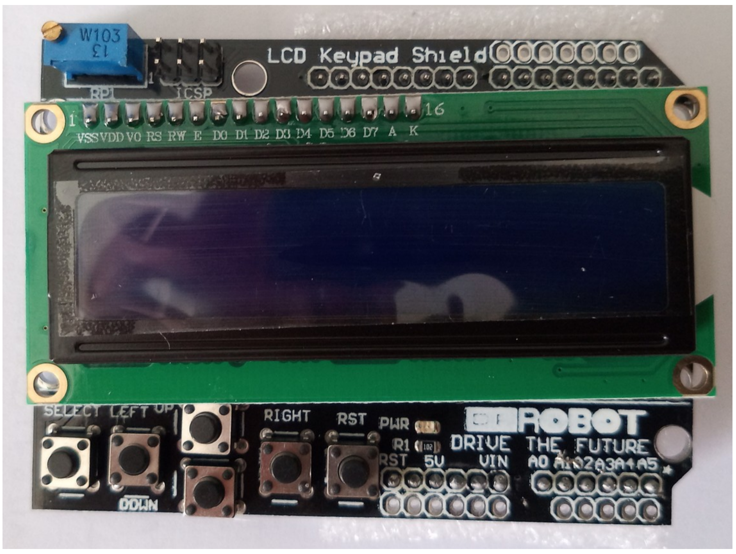
- a computer running Arduino IDE
- 1 pair of wire cutters
- 1 pair of flat-pointy pliers

Figur 7: Examples of flat-pointy pliers

- equipment for soldering
  - Soldering iron
  - Solder
- headphones, mixer or a computer with audio in. See this tutorial for ideas on how to listen: <https://sensorium.github.io/Mozzi/learn/introductory-tutorial/>
- optional: 1 x power adapter for the Arduino (or battery + battery holder) so the OPO can be used without being connected to a computer

And the components:

- 1 x Arduino Uno
- 1 x LCD Keypad shield



Figur 8: LCD Keypad shield

- audio/headphone jack
- wires
- for syncing:
  - 3 x female - male Arduino/electronics patch cables
  - 1 x female - female patch cable

Hardware

We are going to:

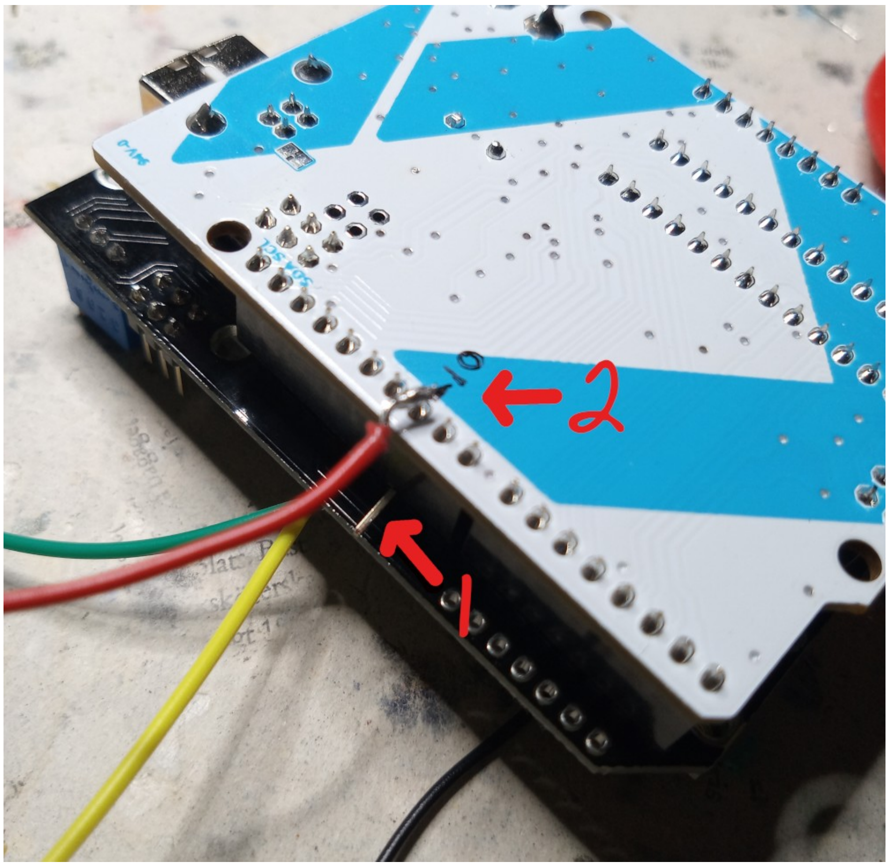
1. attach the LCD Keypad shield
2. attach the Audio jack to be able to hear our synth
3. optionally attach the potentiometers to control the synth

1. LCD Keypad shield

Info: Before attaching the LCD Keypad shield we need to make sure that it does not connect to pin 10 (D10) on the Arduino. *Explanation: D10 is normally used to control the backlight (brightness) on the LCD. But we are going to use D10 for audio.*

To make this work, you have to bend the pin outwards on the shield that goes into D10 (see graphics) on the Arduino. Bend it 90 degrees. Some of the pins might be bent a bit already. Make sure you straighten them out before attaching the LCD Keypad shield. It is easy if you use the flat tweezers.

This image from the back of the Arduino (after attaching the LCD Keypad shield) shows the bent pin of the shield (1). You can also see the audio connection from the Arduino (2) which we are going to fix in the next step.



Figur 9: Back of Arduino with indicators (1) for bent pin, and (2) for pin D10

Now attach the LCD Keypad shield to the Arduino. It is easier if you first place the Arduino on a firm surface (we are later going to do some soldering, make sure that surface can handle a few burn marks.) Make sure *all* the pins of the shield, on both sides, aligns with the ports of the Arduino before pressing it firmly down. *This is an important step – really make sure that all the pins of the shield inserts into the correct connections of the Arduino!*

The LCD Keypad shield with the LCD and the buttons are now connected to the Arduino:

- LCD: D4, D5, D6, D7, D8, D9
- Buttons: A0

Info: The RST (Reset) button resets (restarts) the Arduino but it is not used by this project.

If you want to, you can test the connectivity of the Arduino and the LCD Keypad shield by uploading the code_test_lcd sketch. The screen should display “Hello, world!” and a counter on the second row.

You can also try the “code_test_analog_buttons” example. The Serial Monitor of the Arduino IDE <https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-serial-monitor> should display different numbers depending on which button you press on the LCD Keypad shield.

See the Problem Solving section if you have any problems.

Don’t forget to disconnect the Arduino from your computer before continuing.

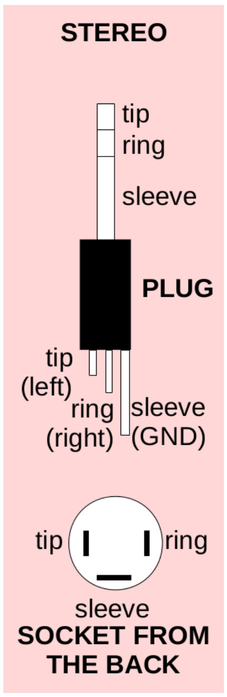
2. Audio jack

The Audio jack is connected to D10 and GND on the Arduino.

Explanation: The sound is created by the Arduino running the Mozzi library <https://sensorium.github.io/Mozzi/>. The Mozzi library normally works with pin D9, but as this connection is used by the LCD Keypad shield, we have to make some configuration changes to the Mozzi library. This is described in the Software section later on, and only involves changing a few lines in the Mozzi configuration code.

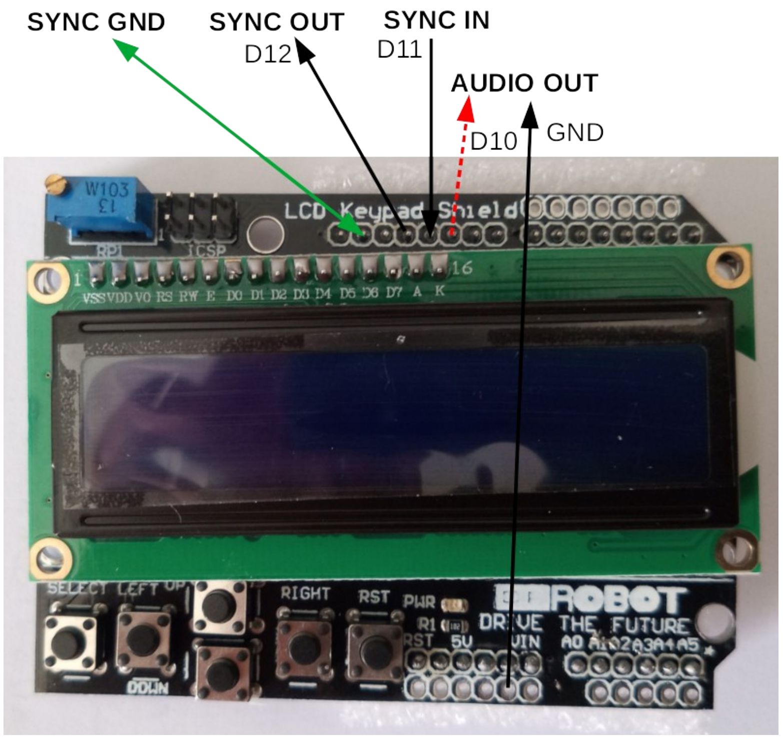
Soldering of wires

1. Solder the *colored* wire (the wire that is neither red or black) to both left (tip) and right (ring) on the Audio jack pins (socket). Use a pair of cutters to remove 1 cm at both ends of the plastic cover. You can pre-lead the ends by applying some lead by the hot tip of the soldering iron andapply it to the exposed cable end to get a better connection.
2. Solder a *black* wire to the sleeve pin (the PLUG graphic is just shown for information):



Figur 10: Schematic of Audio jack

1. Solder the *colored* wire to the D10 of the Arduino. As we already have attached the shield use the D10 solder joint on the back of the Arduino.
2. Solder the *black* wire to Arduino GND which can be found on top of the shield (see picture).



Figur 11: Schematic with indicators of connections to solder

You can now try to download the OscPocketO code to the Arduino. Connect the audio out to a headphone or a speaker. Warning: Remember that the signal level of the OPO might be a bit high (hot) so don’t plug it into your expensive stereo amplifier. Use a cheap active speaker or a set of headphones that you can afford to lose.

Now continue with the Software section on page 21.

3. Optional: Sync in and out and Sync Ground

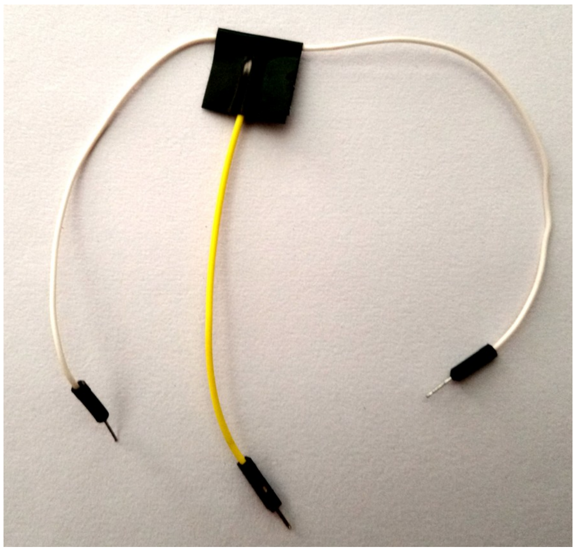
If you have two or more units you might want to synchronize them to play in the same tempo, for example one OPO synth and one OPO drum machine.

For this to work they have to share the same ground (GND) and be connected SYNC OUT to SYNC IN.

Prepare a SYNC signal cable. For two machines an ordinary male-male patch cable is fine. Connect it to SYNC OUT pin D12 on the OPO sending the synchronizing signal, and SYNC IN pin D11 on the OPO receiving the sync signal.

If you have three OPO machines:

- Cut the female-male patch cable in two and solder the oposite end of the female part to D11 (SYNC IN) and the male to D12 (SYNC OUT).
- Cut the female-female patch cable in two and solder one half to GND. It is easiest to select the 2nd GND (where an imaginary "D14" would be).



4. Put it in a box

Info: For durability you should put the OPO into a box and fasten the Audio jack.

Software

Connect your Arduino to your computer running the Arduino IDE.

Install and configure the Mozzi library

Download and install Mozzi using the instructions on the Mozzi site:
<https://sensorium.github.io/Mozzi/download/>

If you need, read more about installing Arduino libraries:
<https://www.arduino.cc/en/Guide/Libraries>

Info: By default Mozzi outputs to D9, but as this pin is used by the LCD Keypad Shield, we have to change this to D10.

In the Mozzi libraries folder <https://support.arduino.cc/hc/en-us/articles/4415103213714-Find-sketches-libraries-board-cores-and-other-files-on-your-computer>, find and open AudioConfigStandardPlus.h in a text editor.

Change “A” to “B” and “B” to “A” on the following four lines so they look like this:

```
// Used internally.  If there was a channel 2, it would be OCR1B.
#define AUDIO_CHANNEL_1_OUTPUT_REGISTER OCR1B
#define AUDIO_CHANNEL_2_OUTPUT_REGISTER OCR1A
```

(...)

```
#define AUDIO_CHANNEL_1_PIN TIMER1_B_PIN // defined in
TimerOne/config/known_16bit_timers.h
#define AUDIO_CHANNEL_2_PIN TIMER1_A_PIN
```

Install the OPO sketch

Download the OPO from <https://oscillator.se/arduino/> (which you probably already have done as you are reading this manual).

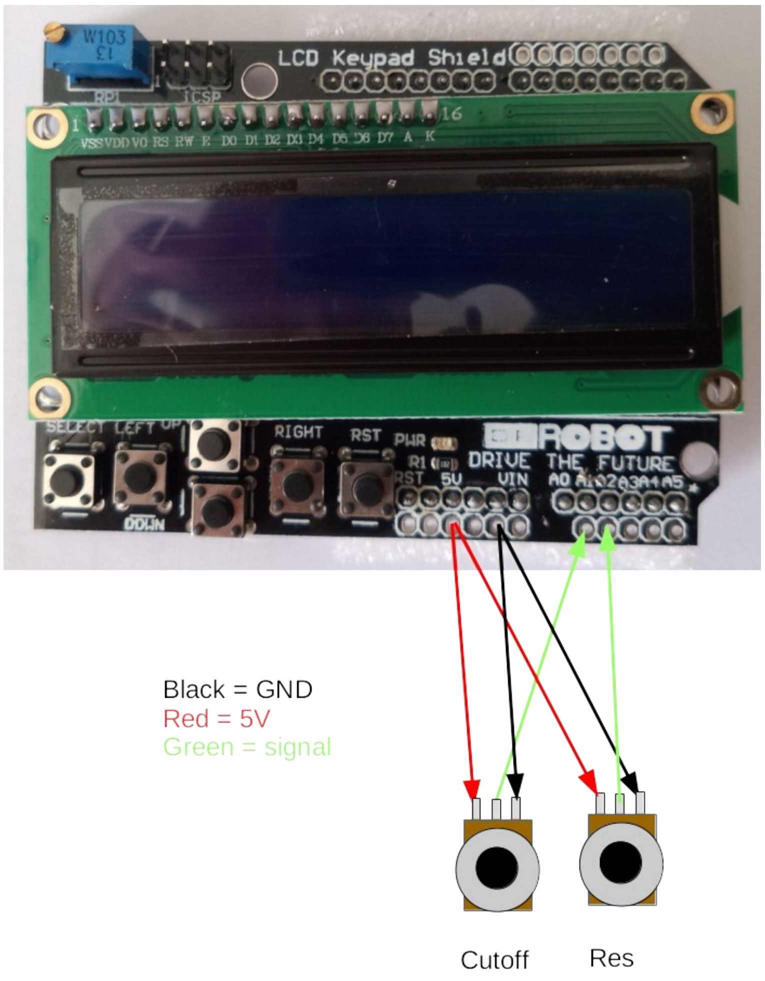
Chose which OPO you would like to run: Synth or Drums. For Synth open the *code_synth/code_synth.ino* or for Drums open *code_drums/code_drums.ino* in the Arduino IDE and upload either to your Arduino.

Expansions

Control the filter with potentiometers

Take two potentiometers. We use two 10k Ohms.

Connect them according to the diagram (A1 and A2).



Figur 12: Connecting the optional potentiometers for Cutoff frequency and Resonance

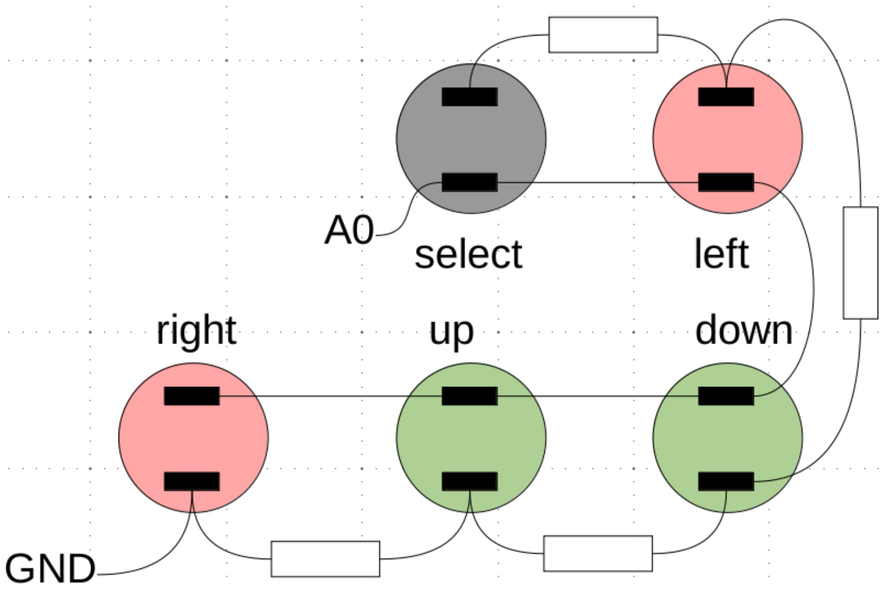
Info: Make sure you set "Filter Mode" to "POTS".

Optional: Building without the LCD Keypad shield

If you don't have a LCD Keypad shield you can build it anyway by replacing the LCD Keypad shield with a DYI keypad or:

- 1 LCD Screen with 16x2 characters (compatible with Hitachi HD44780 driver)
- 5 x pushbuttons (momentary),
- 5 x resistors for buttons (2kΩ or something similar, they must all be of the same value)

Connect the buttons like this using 2k Ω resistors (or other resistors all of the same value):



The buttons are read using one analog in pin (A0) on the Arduino to save digital pins. As resistor values can vary, it is a good idea to measure the voltage when pressing the different buttons and adjust the corresponding values in the *UIHandle()* function:

```
  // is any button pressed?
  if (aUIButtonValue < 900)
  {
    // check which one
    if (aUIButtonValue < 50)
    {
      aUIButton = UI_BUTTON_RIGHT;
    } else if (aUIButtonValue < 150) {
      aUIButton = UI_BUTTON_UP;
    } else if (aUIButtonValue < 300) {
      aUIButton = UI_BUTTON_DOWN;
    } else if (aUIButtonValue < 500) {
      aUIButton = UI_BUTTON_LEFT;
    } else if (aUIButtonValue < 700) {
      aUIButton = UI_BUTTON_SELECT;
    } else {
      aUIButton = UI_BUTTON_NONE;
    }
```

Test the connections using the test code: *code_test_analog_buttons.ino*. See the section Problem solving: The Buttons at the end of the manual if you need help.

Links

- http://tronixstuff.com/2011/01/11/tutorial-using-analog-input-for-multiple-buttons

Connect the LCD to the Arduino. The code that defines which pins to use can be found at the top:

```
// LCD
#define PIN_LCD_D4 4
#define PIN_LCD_D5 5
#define PIN_LCD_D6 6
#define PIN_LCD_D7 7
#define PIN_LCD_RS 8
#define PIN_LCD_EN 9
```

So, D4 on the LCD should go to digital 4 on the Arduino etc.

Problem solving

The screen

You can test the LCD screen using the sketch *code_test_lcd.ino* in the *code_test* folder. It should display "hello, world!" and a ticking time on your screen.

If not you can try the following:

- There is a small blue "box" at the top left of the LCD Keypad shield. This is the contrast control. There is a small screw on it. Try to turn it and see if it helps.
- If you still can't see anything try the advice in this video: <https://www.youtube.com/watch?v=hsJOVG_5pMI>

The buttons

All buttons change a value on pin A0 of the Arduino. It could be that your model of LCD Keypad shield gives different values than ours.

Upload the code_test_analog_buttons.ino from the code_test folder to your OPO.

In the Arduino IDE chose Tools > Serial Monitor. In that window you have a popup where you can select speed. Select 9600.

The monitor should stream values all the time, that is the correct behavior, even if you don't press a button.

Note down the values that appear when you press the different buttons (ignore RST reset button as it just restarts your Arduino). The value fluctuates a bit, this is normal.

These values are detected in the code, in the function *UIHandle()*, which is near line 566 in the code.

It works like this (excerpt from code line 572 onwards):

```
aUIButtonValue = mozziAnalogRead(PIN_BUTTONS);

// is any button pressed?
if (aUIButtonValue < 900)
{
  // check which one
  if (aUIButtonValue < 50)
  {
    aUIButton = UI_BUTTON_RIGHT;
  } else if (aUIButtonValue < 150) {
    aUIButton = UI_BUTTON_UP;
  } else if (aUIButtonValue < 300) {
    aUIButton = UI_BUTTON_DOWN;
  } else if (aUIButtonValue < 500) {
    aUIButton = UI_BUTTON_LEFT;
  } else if (aUIButtonValue < 700) {
    aUIButton = UI_BUTTON_SELECT;
  } else {
    aUIButton = UI_BUTTON_NONE;
  }
```

First the code reads the value on pin A0. This puts a value from 0 to 1023 in the variable aUIButtonValue. This value will be different depending on the pressed button.

If no button is pressed the value will be larger than 900, so we filter that out.

If the value is less than 50 it is the RIGHT button.

If the value is less than 150 it is the UP button.

If the value is less than 300 it is the DOWN button.

If the value is less than 500 it is the LEFT button.

If the value is less than 700 it is the SELECT button.

This means that the value, when you press the LEFT button, must be between 300 and 500.

Now, your shield could give different values to our LCD shield, so you might have to change the values in the code.