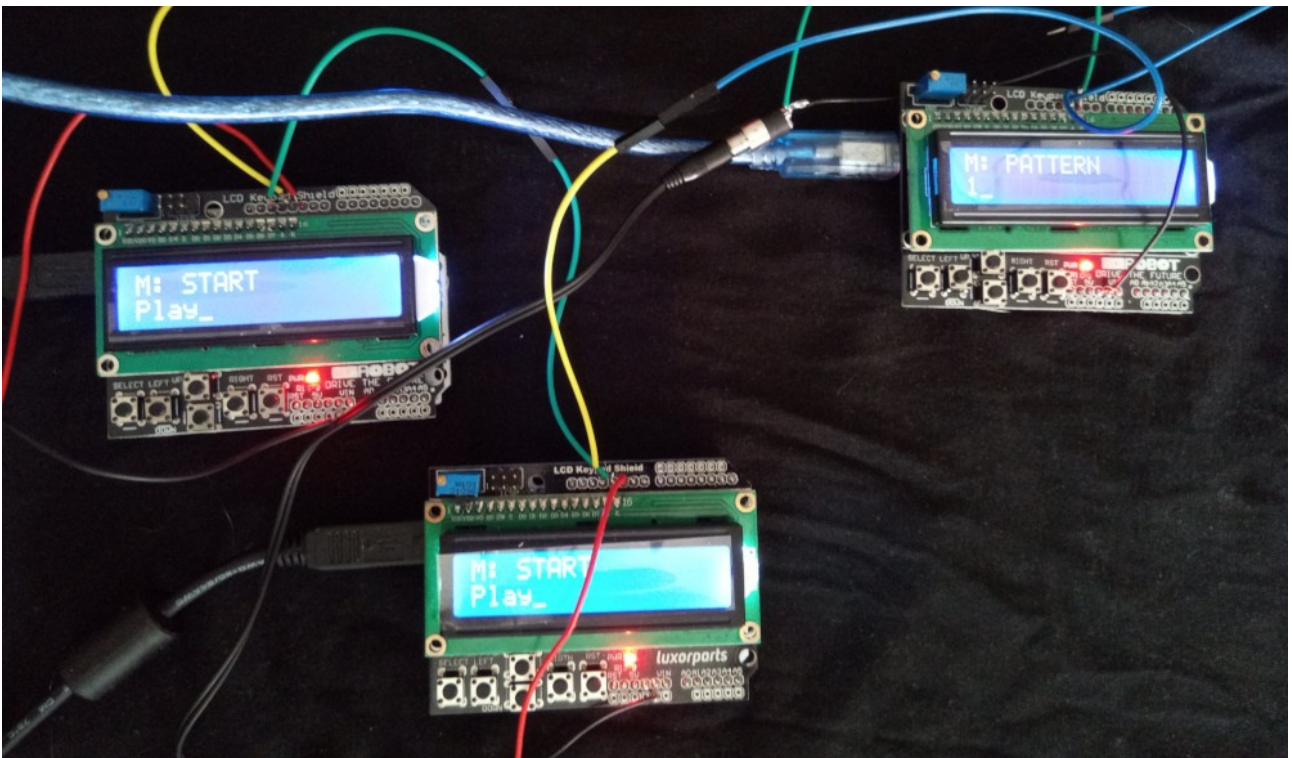


# OSCPOCKETO

Project: OscPocketO  
Author: Staffan Melin, [staffan.melin@oscillator.se](mailto:staffan.melin@oscillator.se)  
License: GNU General Public License v3.0  
Version: 20240904  
Project site: <http://oscillator.se/arduino>



## Introduction

Welcome to the OscPocketO - Arduino Pocket Synth!

The OscPocketO (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 (<https://sensorium.github.io/Mozzi/>)!

This guide assumes you know how to connect, edit and send sketches to an Arduino.

OPO is currently two different machines: OPO Synth and OPO Drums.

## Synth features

The OPO Synth features:

- a 16 step sequencer

- 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
- an optional second detunable oscillator
- the ability to save and load synth settings and patterns to EEPROM
- functions to create patterns
- a play mode for soloing

## **Drums features**

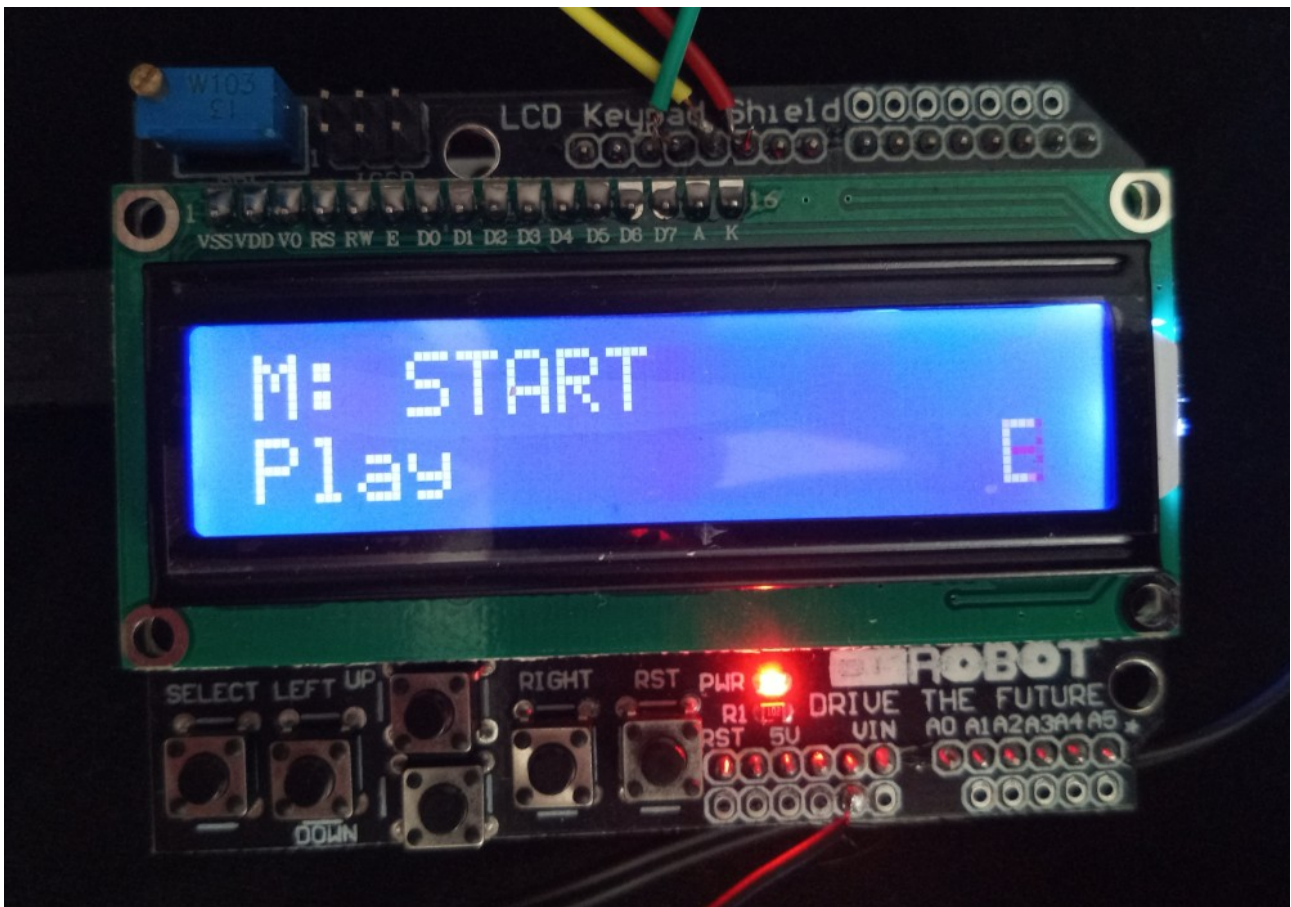
The OPO Drums features:

- 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 soloing

## Table of Contents

Introduction.....	1
Synth features.....	1
Drums features.....	2
How to use it.....	4
Synth.....	4
Drums.....	6
Syncing several OPO machines.....	8
1. Setup.....	8
2. Play.....	8
How to build it.....	9
Equipment.....	10
Hardware.....	11
1. LCD Keypad shield.....	11
2. Audio jack.....	12
3. Sync in and out and Sync Ground.....	13
4. Put it in a box.....	13
Software.....	14
Install and configure the Mozzi library.....	14
Install the OPO sketch.....	14
Expansions.....	15
Control the filter with potentiometers.....	15
Building without the LCD Keypad Shield.....	16
Problem solving.....	18
The screen.....	18
The buttons.....	18

## How to use it



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.

*The Arduino built in LED blinks every time the OPO plays a note.*

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

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

## Synth

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).

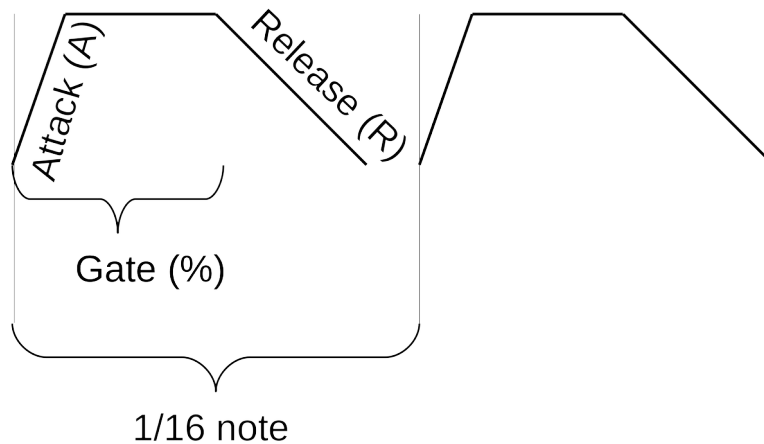
**PATTERN.** Select the current pattern.

**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.



**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 “Expansions” later 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.

**TOOLS.** Utilities. 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.

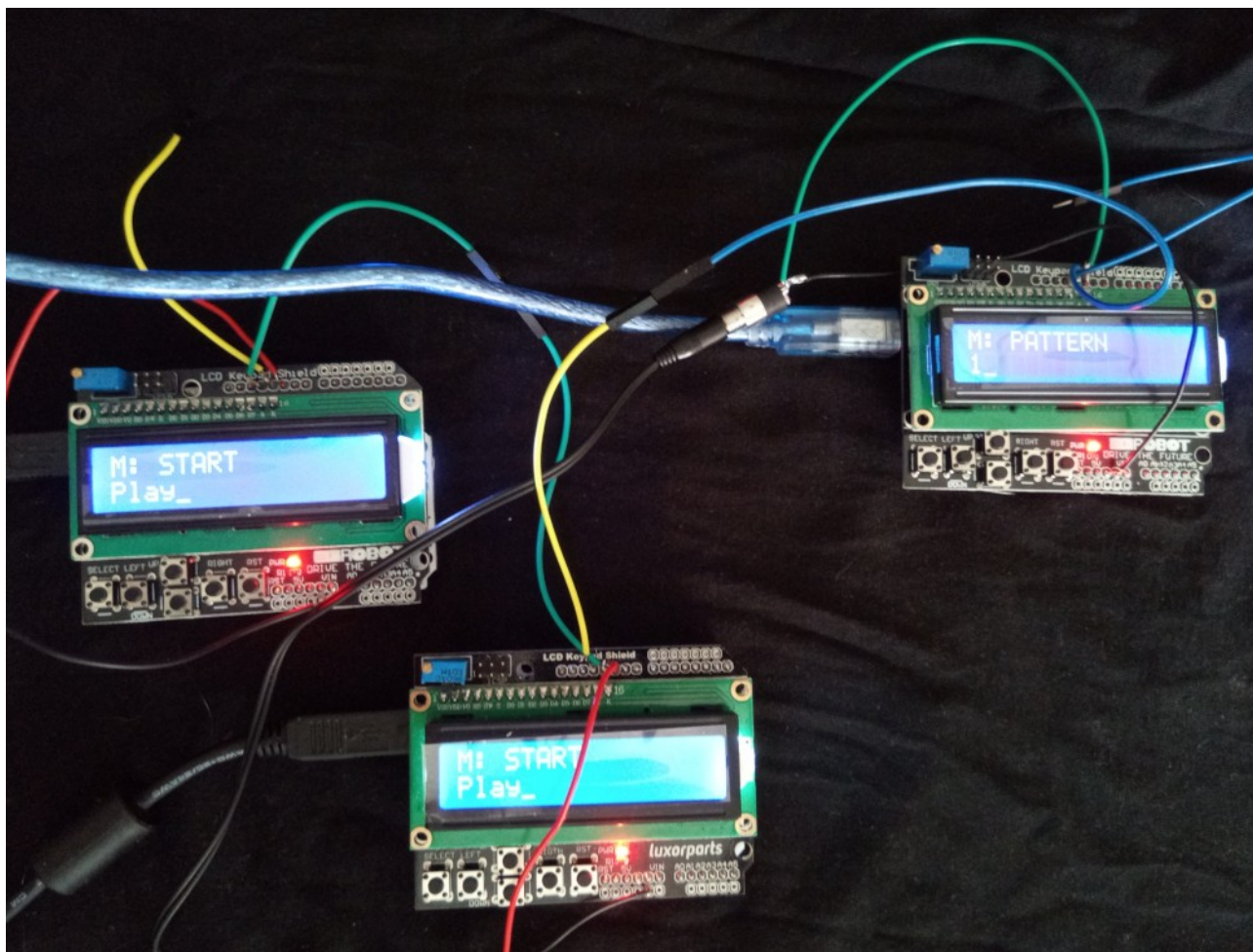
**PLAY.** Solo mode. LEFT = Kick, UP = Snare, DOWN = Tom and RIGHT = Crash.

**TOOLS.** Utilities. 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



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.)

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 OPD:s.

## 2. Play

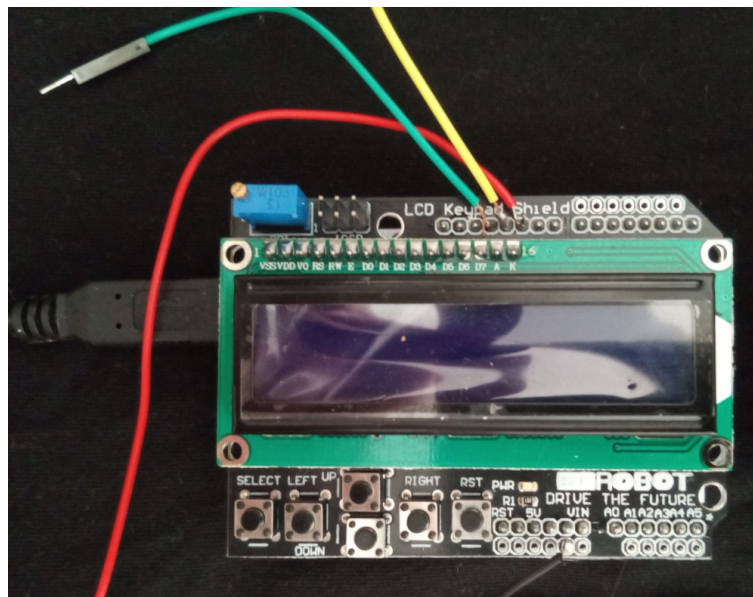
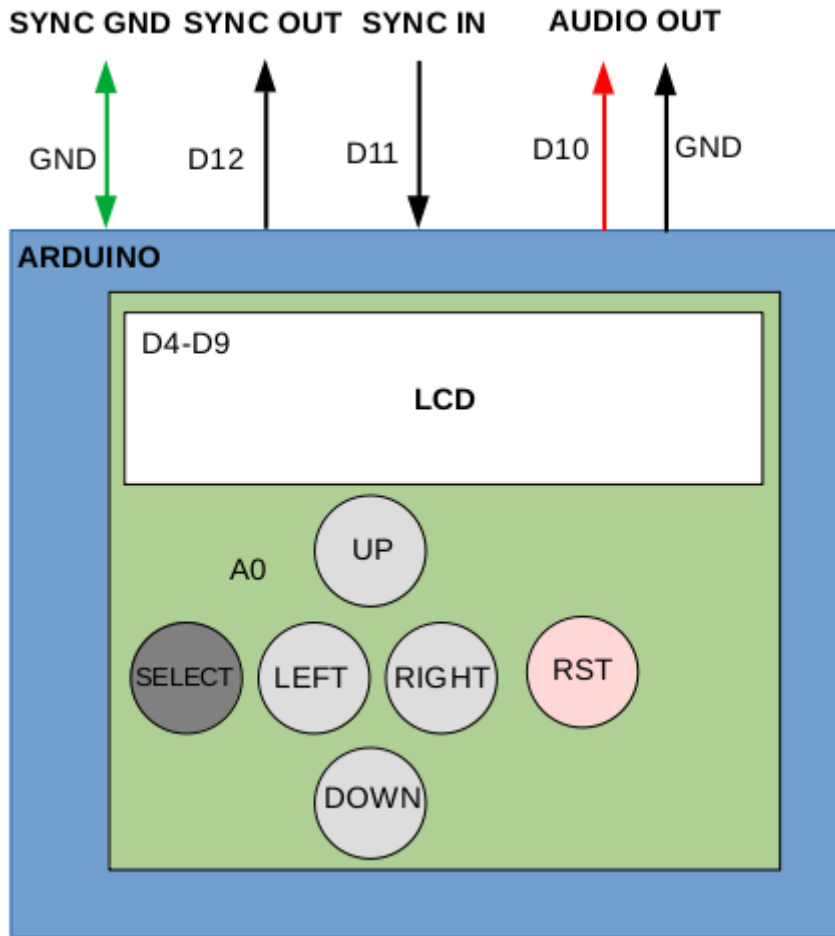
Conductor. Start: Play.

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



## How to build it

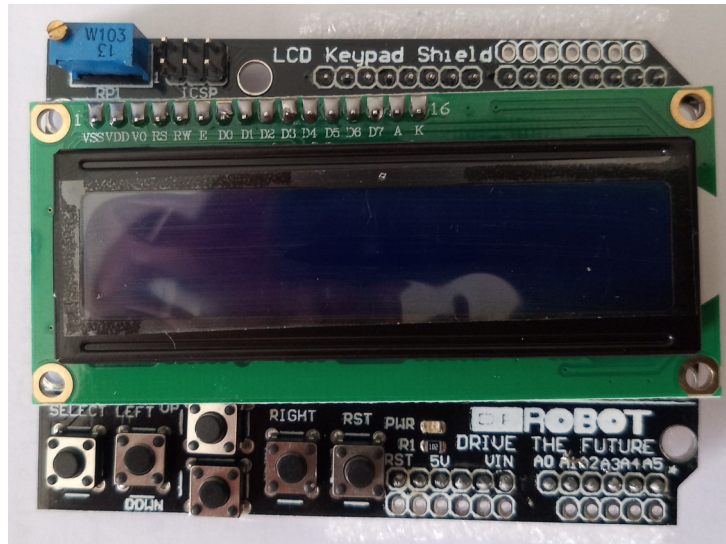
System overview:



(SYNC GND not attached yet)

## Equipment

- a computer running Arduino IDE
- 1 x Arduino Uno
- 1 x LCD Keypad shield



- audio/headphone jack
- wires
- for syncing:
  - 3 x female - male Arduino/electronics patch cables
  - 1 x female - female patch cable
- 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/>)
- 1 x power adapter for the Arduino (or battery + battery holder) so the OPO can be used without being connected to a computer
- equipment for soldering

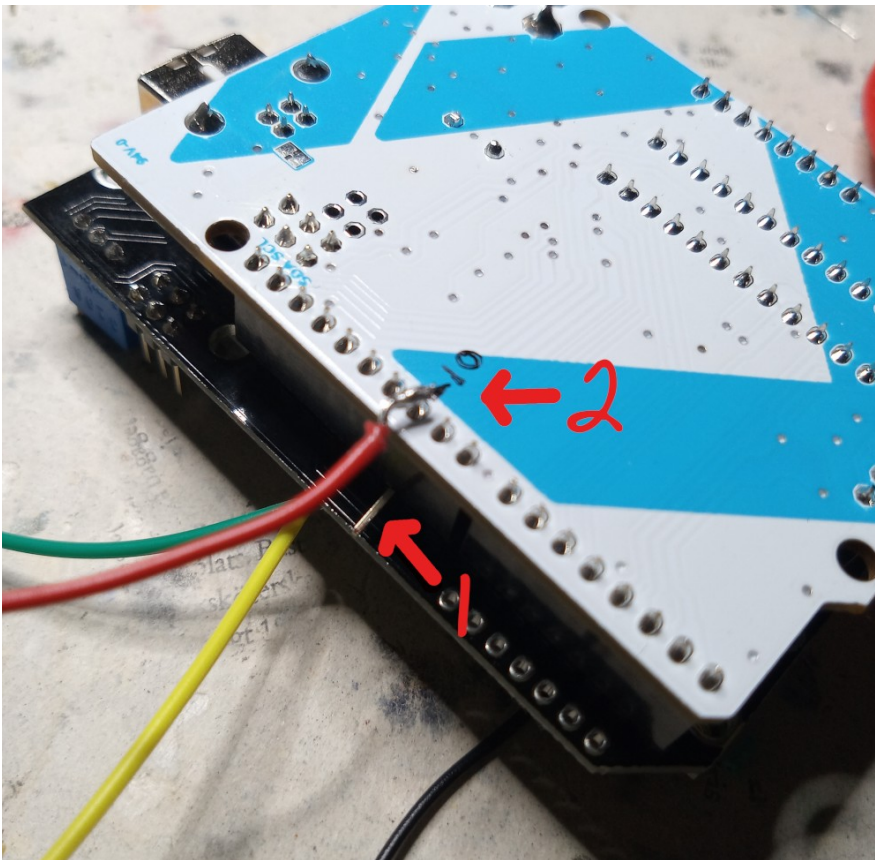
# Hardware

## 1. LCD Keypad shield

Before attaching the LCD Keypad shield we have to make sure that it is not connected to pin 10 (D10) on the Arduino. 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 on the shield that goes into D10 on the Arduino. Bend it 90 degrees.

This image from the back of the Arduino shows the bent pin of the shield (1) and the audio connection from the Arduino (2) which we are going to fix in the next step.



Now attach the LCD Keypad shield.

The LCD and the buttons are connected to the Arduino by the Shield:

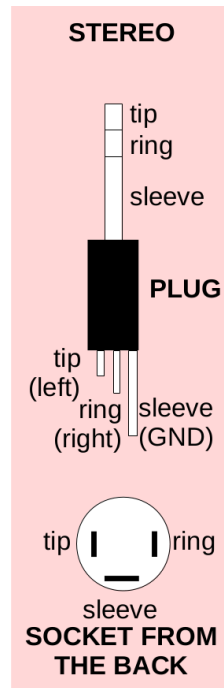
- LCD: D4, D5, D6, D7, D8, D9
- Buttons: A0

The RST (Reset) button resets (restarts) the Arduino and is not used by the code.

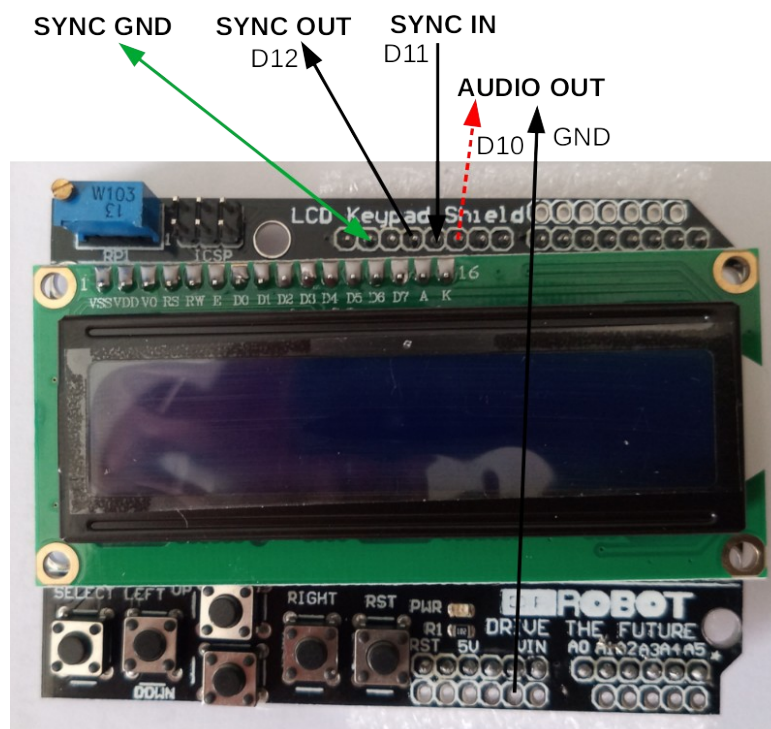
## 2. Audio jack

The audio jack is connected to D10 and GND on the Arduino. The Mozzi library normally works with D9, but as this connection is used by the LCD Keypad shield, we have to make some changes to the Mozzi library. This is described in the Software section later on.

Solder a colored wire to both left and right on the audio jack (socket) and solder a black wire to the sleeve (the plug is just shown for information):



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. Solder the black wire to Arduino GND which can be found on top of the shield.

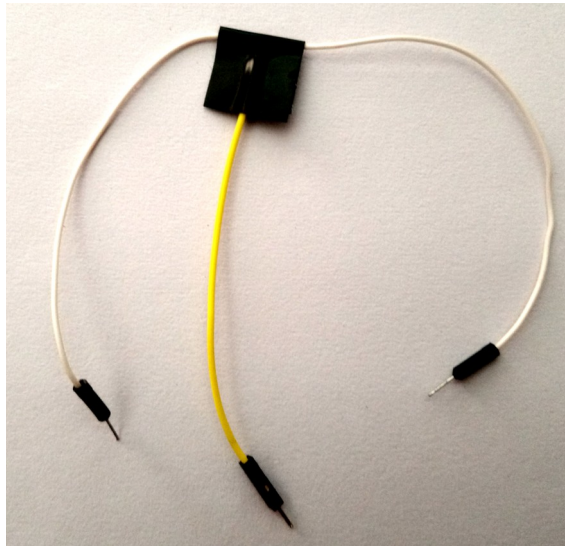


### 3. Sync in and out and Sync Ground

Cut the female-male patch cable in two and solder the female to D11 (SYNC IN) and the male to D12 (SYNC OUT).

Cut another 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).

Prepare a SYNC GND-cable. For two machines an ordinary male-male patch cable is fine. If you have three machines prepare something like this:



### 4. Put it in a box

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

Inside the Arduino IDE, select Tools > Manage Libraries and install:

- LiquidCrystal
- Mozzi (version >= 2)

If you need, read more about installing Arduino libraries:

<https://www.arduino.cc/en/Guide/Libraries>.

### Install the OPO sketch

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

Chose which OPO you would like to try: Synth or Drums. Open the code\_synth/code\_synth.ino och code\_drums/code\_drums.ino in the Arduino IDE and upload them to your Arduino.

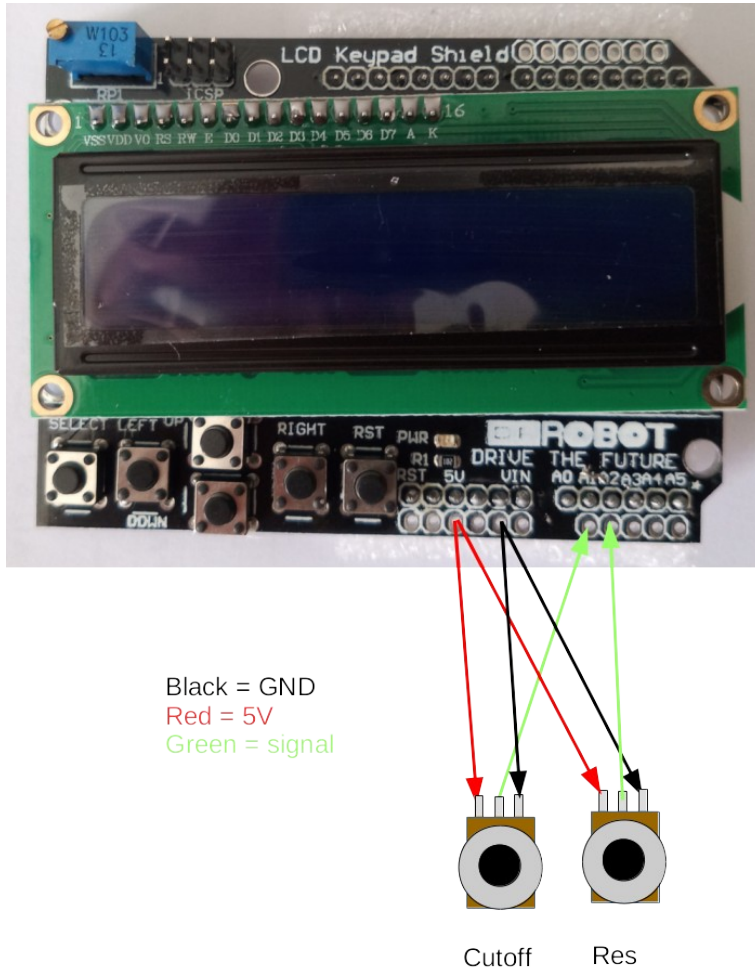


## Expansions

### Control the filter with potentiometers

Take two pots. I used two 10k Ohms.

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



Make sure you set “Filter Mode” to “POTS”.

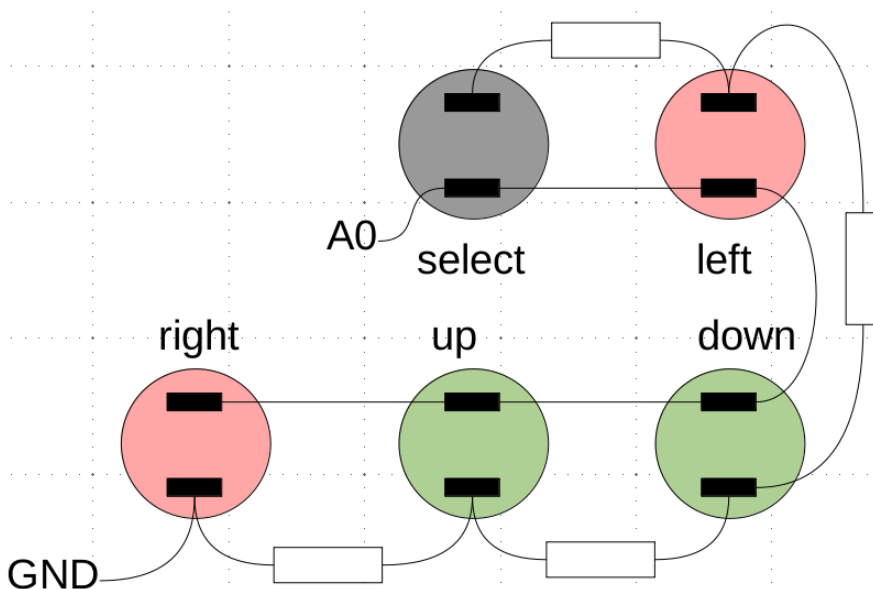


## Building without the LCD Keypad Shield

If you don't have an LCD Keypad Shield you can build it anyway by replace the Shield with:

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

Connect the buttons like this using 2k  $\Omega$  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 program 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](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 mine.

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 = mozziaAnalogRead(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 my LCD shield, so you might have to change the values in the code.