

OscDigiSeq

4+1-channel CV/Gate sequencer

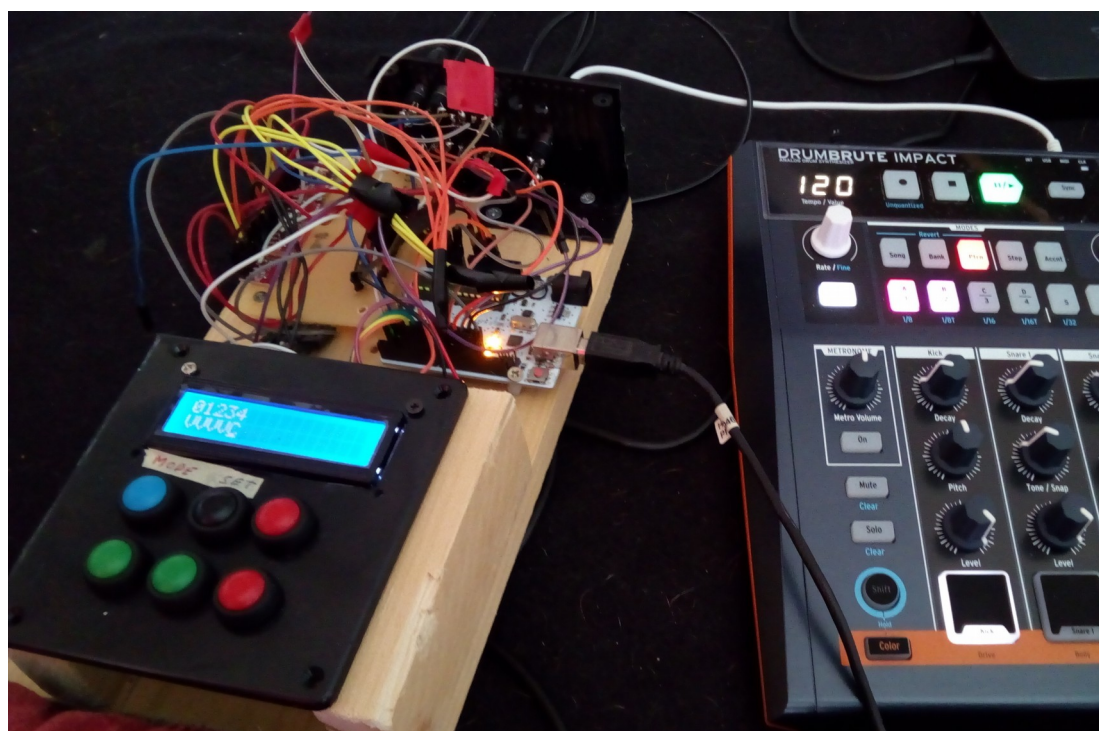
Project: OscDigiSeq

Author: Staffan Melin, staffan.melin@oscillator.se

License: GNU General Public License v3.0

Version: 20200120

Project site: <http://www.oscillator.se/arduino>



Contents

Introduction.....	2
How to use it.....	3
How to build it.....	5
Equipment.....	7
LCD.....	8
Buttons.....	9
DACs.....	12
Sockets.....	14
Arduino connections.....	16
Code.....	17
Note data.....	18
Modes.....	19
Ideas for improvement.....	20

Introduction



The OscDigiSeq is a 4+1 channel analog CV/Gate sequencer based on an Arduino Uno R3.

It has 4 CV/Gate channels that operate from 0 to +5V to drive 4 analog synthesizers. It also has a gate (clock) channel for synchronizing with an analog drum machine.

Every channel (voice) has 2 patterns consisting of 16 notes ranging from 0-60 (5 octaves).

The software operates using a simple GUI with a 16x2 LCD display and 6 buttons that allows you to:

- select pattern or turn channel (voice) off
- edit the notes (or clock on/off for the clock channel)

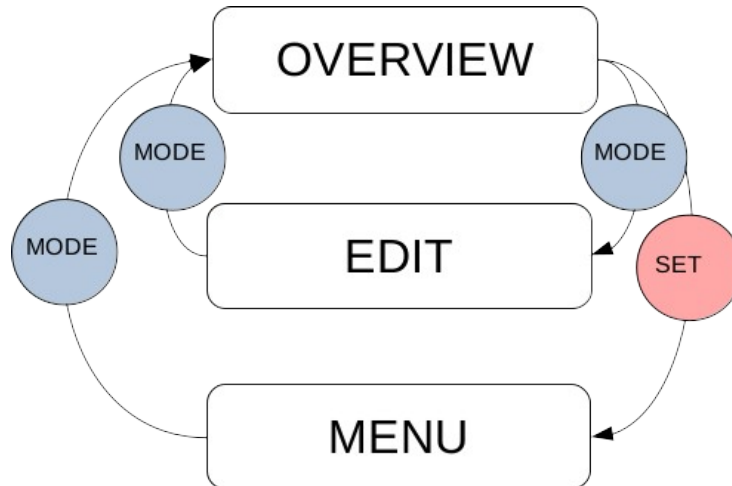
There is also a utility menu that allows you to

- start and stop the sequencer [A/O]
- set the tempo (from 0 to 999) [T]
- edit the gate length (from 0-99%) for each channel [G]
- transform a pattern (shift it left/right and transpose it) [R]
- save the pattern as Arduino code [s] or to and from the Arduino EEPROM [S/L]
- copy the current pattern to the next [Y]
- tune all connected devices [U]
- enter drone mode [D]
- edit pattern chain [c]
- play pattern chain [C]

How to use it

The sequencer is off at the start. You can start and stop it in the utility menu.

You move between Overview and Edit with the MODE button. To go to the utility menu press SET in the Overview mode.



Overview

This mode displays all 5 voices and the current pattern or off (-). Move with LEFT and RIGHT and change pattern or turn off using UP and DOWN. Press MODE to go to the Edit mode. Press SET to go to the utility menu.

Edit

This mode displays the pattern of a voice with one note per column. Move with LEFT and RIGHT and change the value with UP and DOWN. Press SET to turn note on/off. Press MODE to go back to the Overview mode.

Utility menu

Select choice using LEFT and RIGHT. Activate with SET. Press MODE to go back to OVERVIEW mode.

Start [A]

Start the sequencer (from the start of the current pattern).

Stop [O]

Stop the sequencer and stop all devices by setting the gates to LOW.

Tempo [T]

Change the value with UP and DOWN.

Gate [G]

This mode displays the gate time as a percentage of a 1/16th note of all 5 voices. Move with LEFT and RIGHT and change the value with UP and DOWN.

Transform [R]

This mode displays the pattern of the current voice with one note per column. Shift the pattern with LEFT and RIGHT and transpose with UP and DOWN. Turn off all notes in the pattern by pressing SET.

Save as code [s]

Save the pattern as Arduino code for insertion into the setup() function.

Save to and load from the Arduino EEPROM [S/L]

Saves all voice and pattern data to the EEPROM of the Arduino. Load it back.

Copy pattern [c]

Copy the current pattern of the current voice to the next pattern (ie 1->2, 2->1).

Tune [U]

Play note 12 on all devices. Exit tuning by pressing MODE.

Drone [D]

Enter drone mode. This mode turns on all gates and sets all voices to a fixed note/voltage. This voltage can be edited for all four channels using the UP, DOWN, LEFT and RIGHT keys.

The values are in hex (from 0 to FFF).

Exit drone mode by pressing the MODE button.

Edit pattern chain [c]

You can chain the patterns you have made using this mode. Maximum number of chained patterns are 16.

Press SET to turn a pattern off or on (default 1). The chain ends at the first - (pattern off).

Play pattern chain [C]

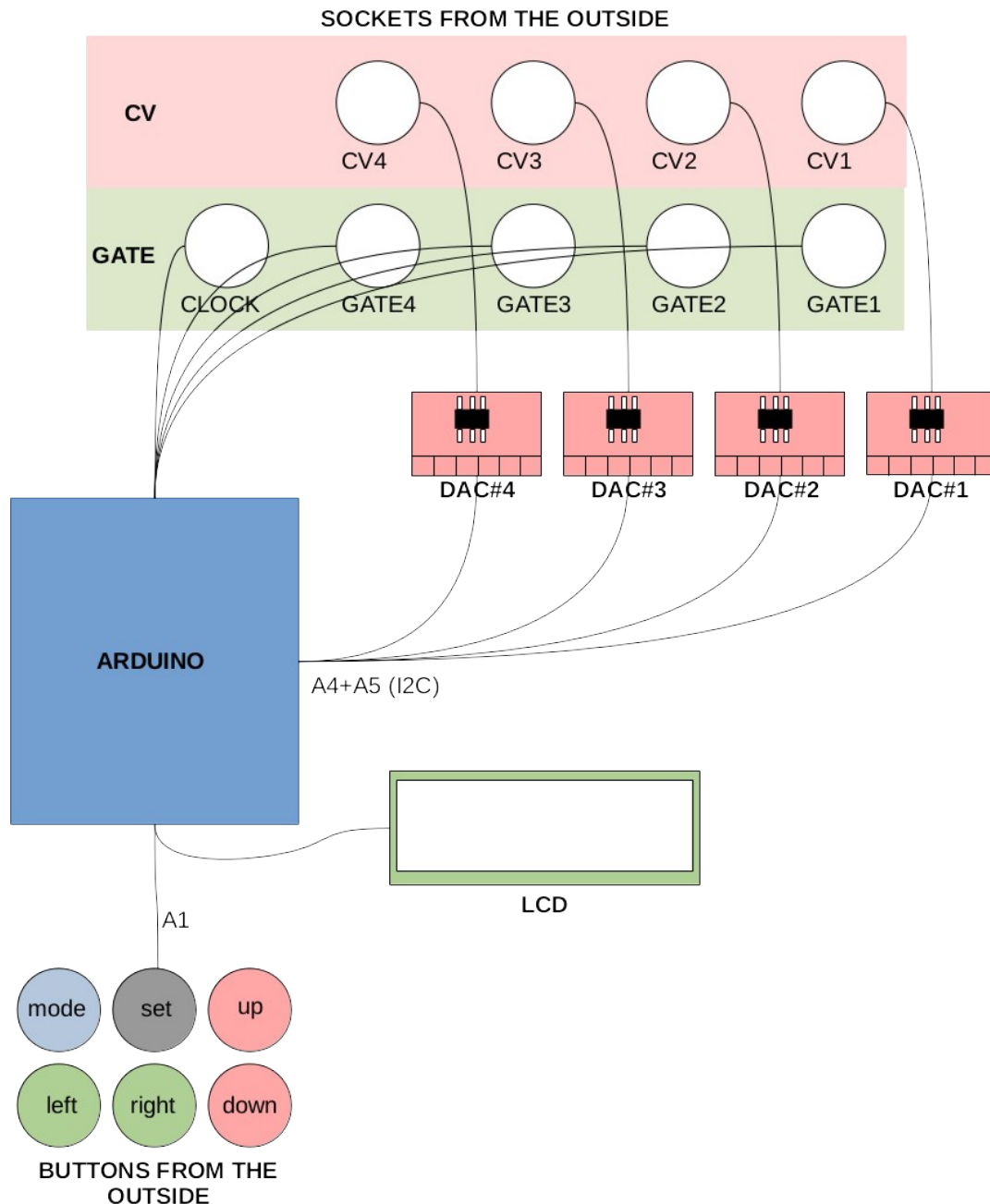
Play the pattern chain for all voices. The pattern chain is displayed. Exit by pressing SET.

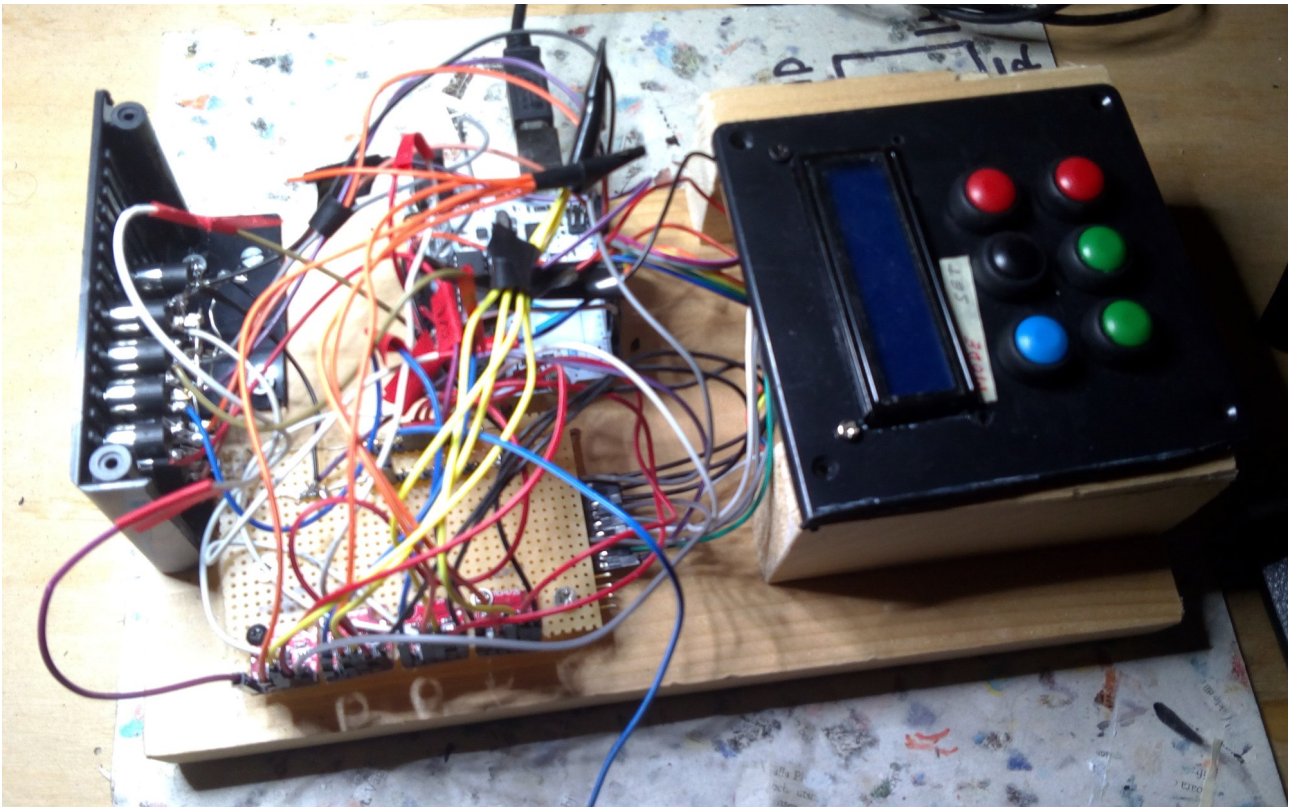
How to build it

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

It is also useful to create some sort of common connection for +5V and GND on the prototyping/perf board. I tried to fasten header strips but they broke during the project, so I ended up just soldering a lot of wires together.

System overview:





Equipment

- 1 x Arduino Uno R3
- 4 x DAC MCP4725
- 6 x pushbuttons (momentary)
- 1 LCD Screen with 16x2 characters (compatible with Hitachi HD44780 driver)
- 9 x 3.5mm mono audio sockets (female)
- 5 x resistors for buttons ($2k\Omega$)
- prototyping board/perf board
- connection wires

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



LCD

Connect the LCD:

LCD pin	Use	Connects to
16	LED-	Arduino GND
15	LED+	Arduino +5V
14	7	Arduino D7
13	6	Arduino D6
12	5	Arduino D5
11	4	Arduino D4
10		not used
9		not used
8		not used
7		not used
6	EN	Arduino D9
5	R/W	Arduino GND
4	RS	Arduino D8
3	Contrast	Arduino GND
2	VDD	Arduino +5V
1	VSS	Arduino GND

Test the connections using the test code: code_test_lcd.ino.

Links

- <https://www.arduino.cc/en/Reference/LiquidCrystal>

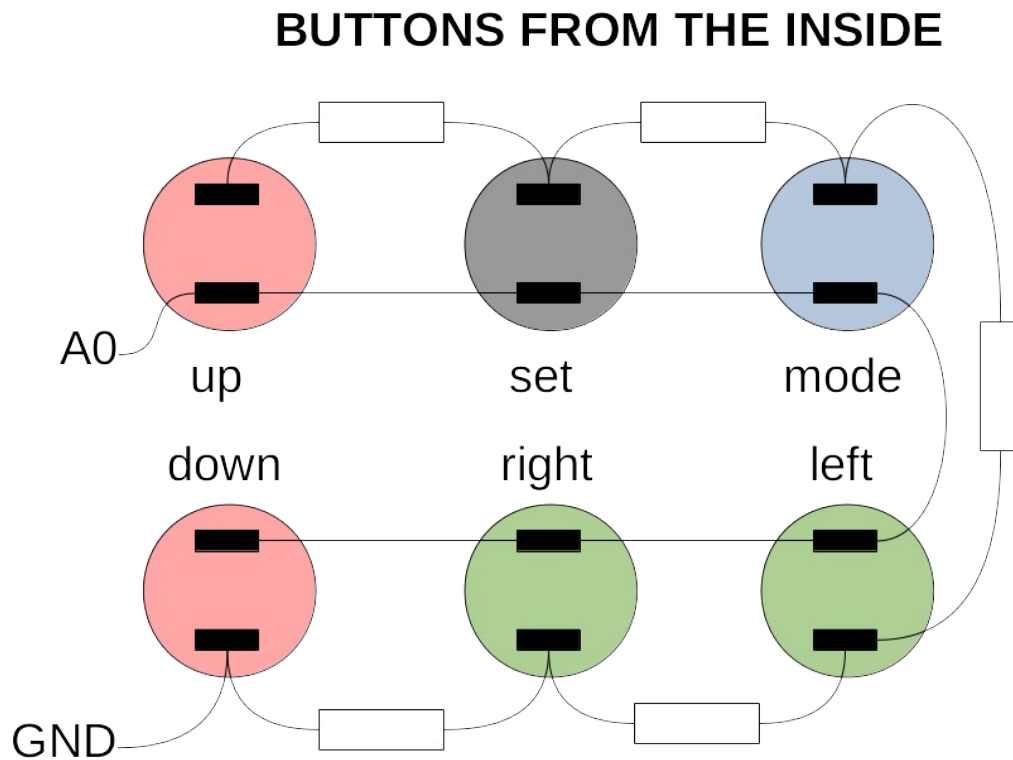
Buttons

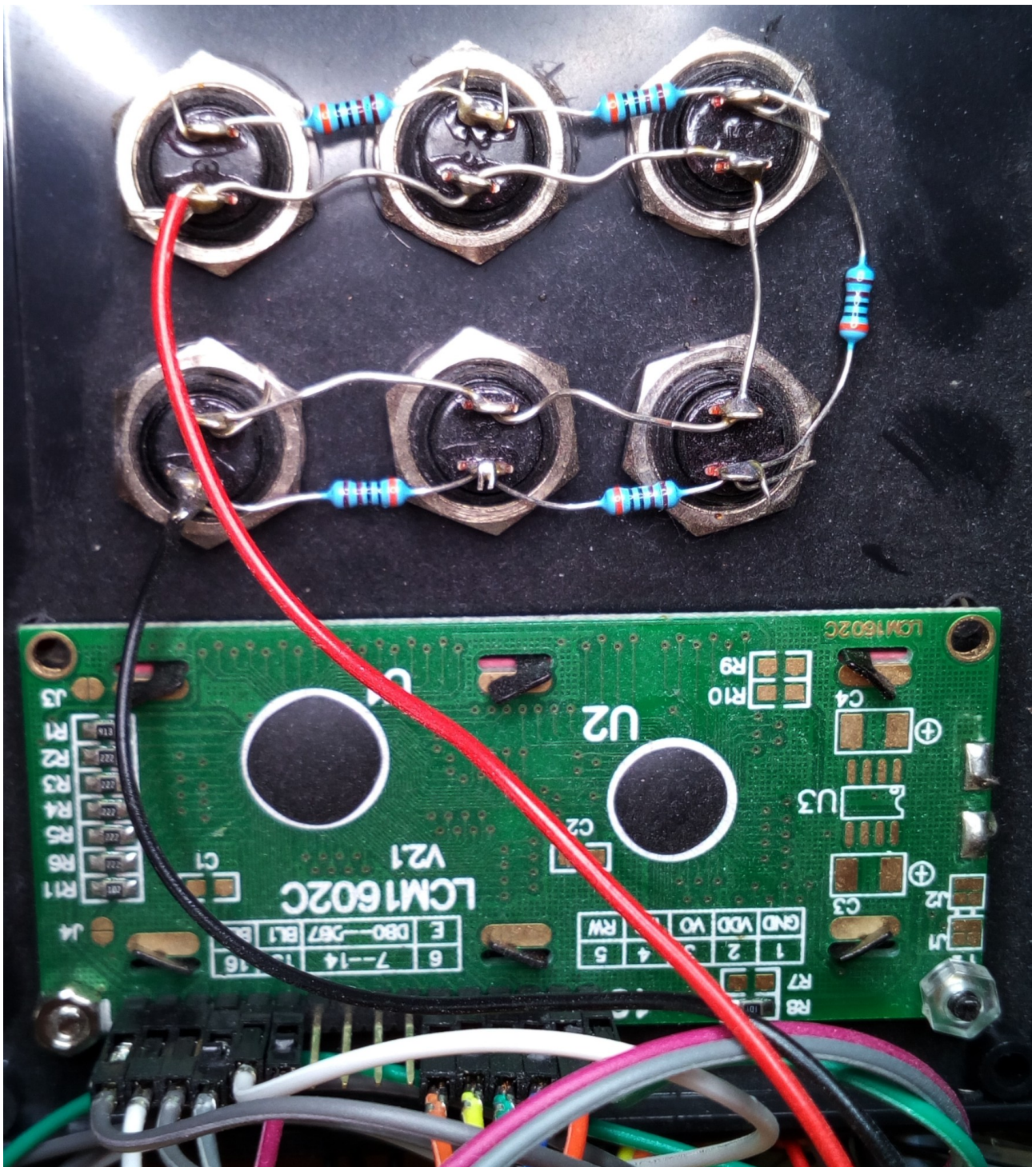
The 6 momentary buttons are used for the following functions:

- up (red)
- down (red)
- left (green)
- right (green)
- set (black)
- mode (blue)

You can of course use any colors that you like.

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





The buttons are read using one analog in pin (A1) 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 `getButton()` function:

```
byte getButton(void) {  
    byte button;  
  
    int buttonValue = analogRead(PIN_BUTTONS);  
  
    if (buttonValue < 20) {  
        button = BUTTON_MODE;  
    } else if (buttonValue < 110) {  
        button = BUTTON_SET;  
    } else if (buttonValue < 170) {  
        button = BUTTON_UP;  
    } else if (buttonValue < 225) {  
        button = BUTTON_DOWN;  
    } else if (buttonValue < 270) {  
        button = BUTTON_RIGHT;  
    } else if (buttonValue < 320) {  
        button = BUTTON_LEFT;  
    } else {  
        button = BUTTON_NONE;  
    }  
}
```

Test the connections using the test code: code_test_analog_buttons.ino.

Links

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

DACs

The OscDigiSeq 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.

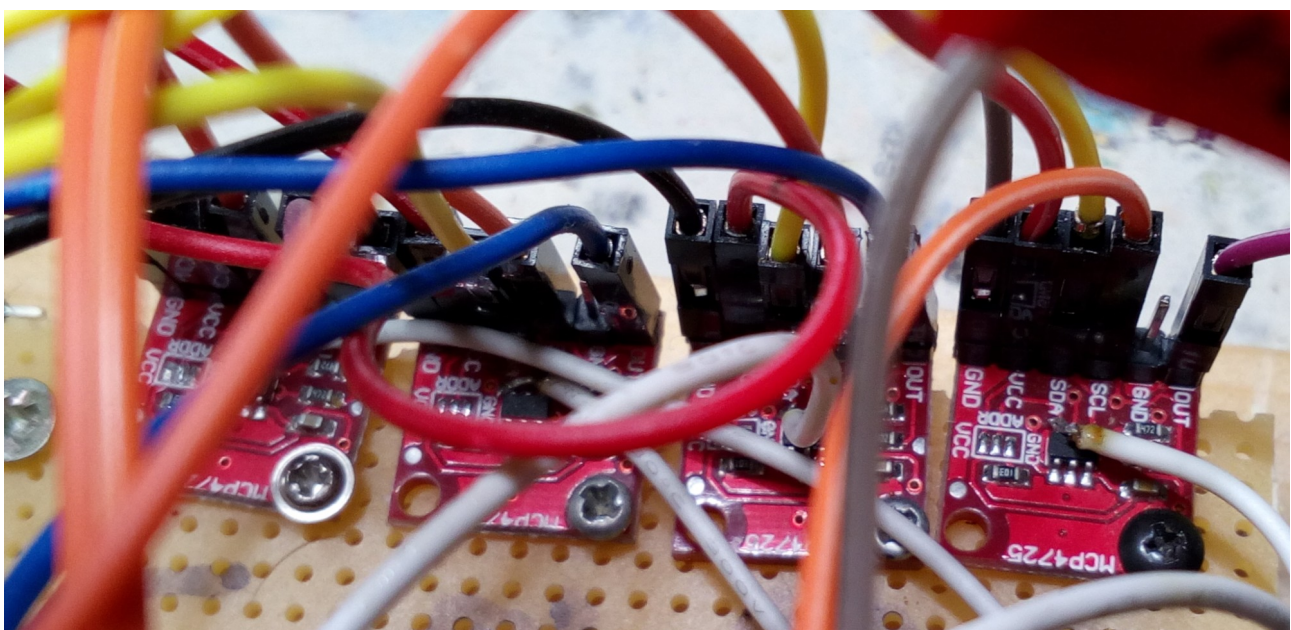
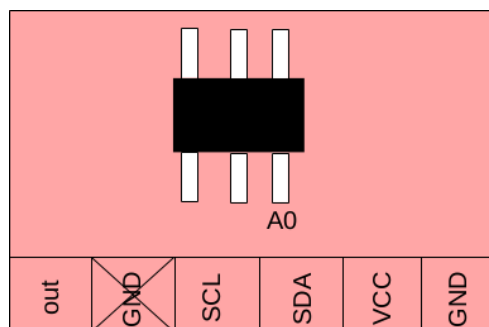
The Arduino controls the Digital to Analog Converters (DACs) using the I2C protocol. For this to work the DAC boards must have unique addresses.

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 4 synthesizers 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 so I had to solder a wire directly to the A0 leg of the DAC chip.



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
DAC #4		
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

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

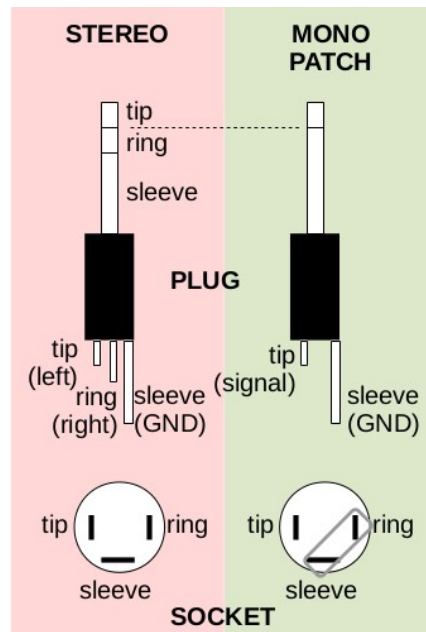
I have read that you need to disable the pullup resistors on all but one DAC if you use more than one, but I didn't do this (<https://learn.sparkfun.com/tutorials/mcp4725-digital-to-analog-converter-hookup-guide>).

Links

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

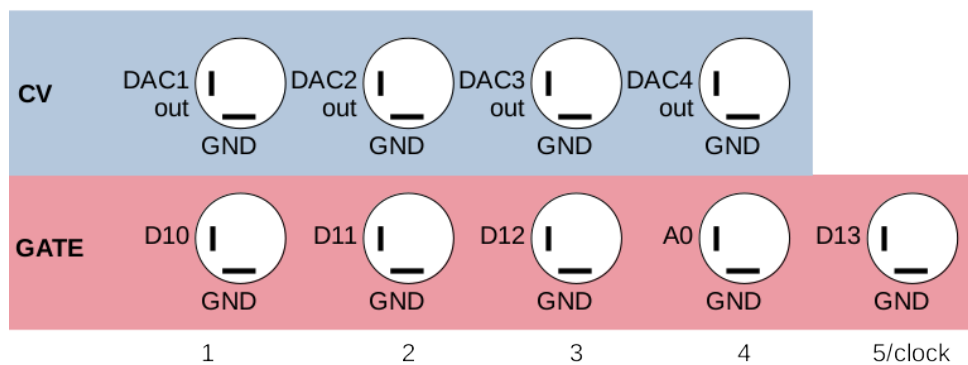
Sockets

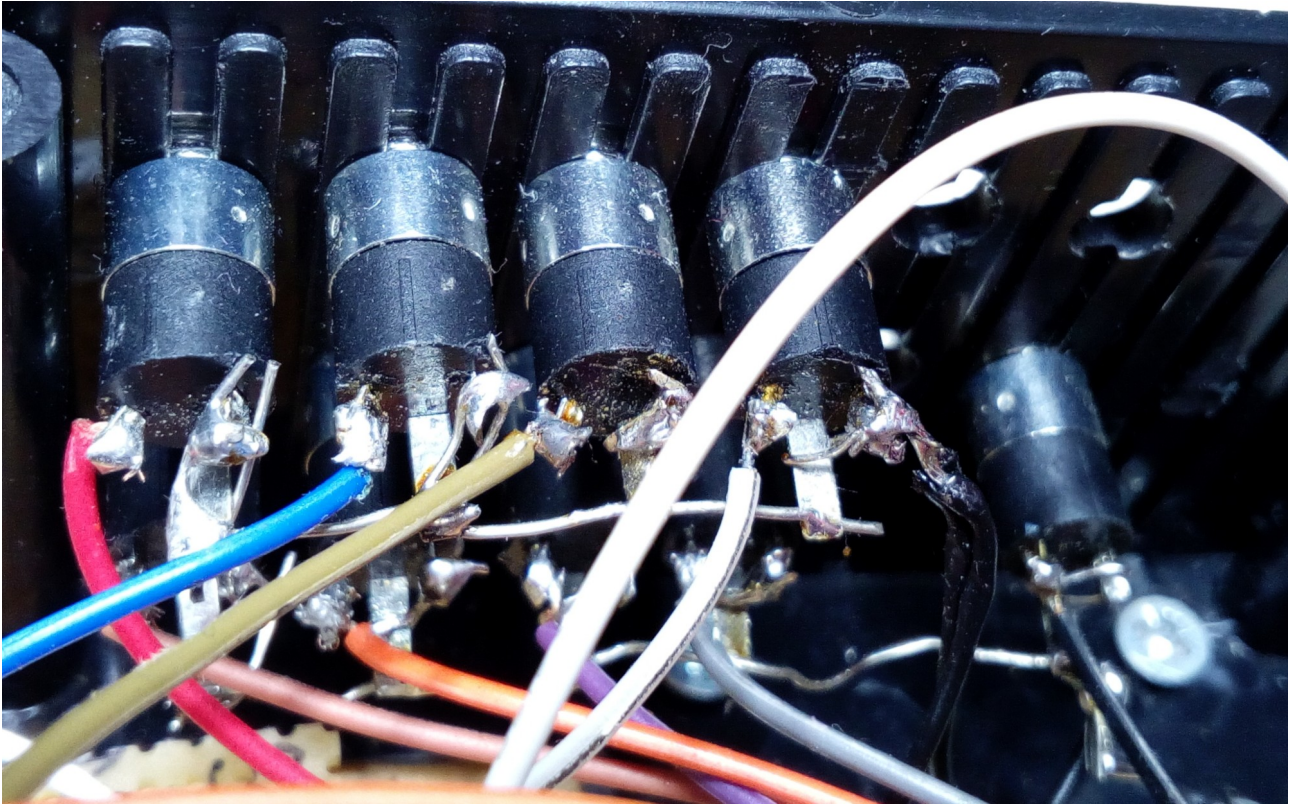
You need 9 x 3.5mm female mono sockets. I used stereo sockets and shorted the ring and the sleeve.



Connect the sockets:

SOCKETS FROM THE INSIDE





Arduino connections

Arduino pin	Use	Connects to
D0	not used	
D1	not used	
D2	CV #1 DAC select	DAC #1 A0 pin
D3	CV #2 DAC select	DAC #2 A0 pin
D4	LCD 4	LCD #11
D5	LCD 5	LCD #12
D6	LCD 6	LCD #13
D7	LCD 7	LCD #14
D8	LCD RS	LCD #4
D9	LCD EN	LCD #6
D10	Gate #1	Gate #1 socket tip
D11	Gate #2	Gate #2 socket tip
D12	Gate #3	Gate #3 socket tip
D13	Clock	Clock socket tip
A0	Gate #4	Gate #4 socket tip
A1	buttons	buttons
A2	CV #3 DAC select	DAC #3 A0 pin
A3	CV #4 DAC select	DAC #4 A0 pin
A4	DAC SDA	SDA port of all 4 DACs
A5	DAC SCL	SCL port of all 4 DACs

Code

The code should be fairly easy to understand, and I have made some comments in it.

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

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

The code also has some demo patterns in setup(). You can find the notes between the lines

```
// demo notes BEGIN
```

and

```
// demo notes END
```

I recommend that you at least keep the last part

```
// drums/click (channel: MAX_NOTES - MAX_CLOCKS)
seqVoiceStatus[4] = NOTE_ON;
for (byte i = 0; i < MAX_NOTES; i++) {
    seqValue[4][i] = 128;
    seqStatus[4][i] = NOTE_ON;
}
```

if you want to synchronize a drum machine with channel 5 (clock), as it contains a pattern of 1/16ths. Set your drum machine to 1 step per 1/16.

Note data

The sequencer works with 4 CV/Gate voices and an additional clock voice. These are handled in nearly identical ways.

All voice and pattern data are stored in arrays:

```
byte seqVoicePattern[MAX_VOICES];
```

Current pattern for each voice (PATTERN_1/PATTERN_2 or PATTERN_OFF).

```
byte seqValue[MAX_VOICES][MAX_PATTERNS][MAX_NOTES];
```

This holds the patterns (note data) for each pattern for each voice, consisting of analog values from 0-NOTE_MAX. The values are not used for the clock voice as this channel only outputs a gate signal.

```
byte seqStatus[MAX_VOICES][MAX_PATTERNS][MAX_NOTES];
```

This holds the corresponding status of each note. It can be either on (NOTE_ON) or off (NOTE_OFF).

```
byte seqGatePin[MAX_VOICES] = {PIN_GATE1, PIN_GATE2, PIN_GATE3, PIN_GATE4, PIN_CLOCK1};
```

This holds the Arduino pin numbers for each Gate output.

```
byte seqGatePercent[MAX_VOICES] = {50, 50, 50, 50, 50};
```

This holds the percent value (of 1/16, from 0-GATE_MAX) for each voice. Default is 50%.

Modes

The different modes of the GUI are handled through a simple state machine. The current state is kept in the `uiMode` variable.

- `UI_MODE_*`

The GUI is handled in the latter half of the `loop()`.

Current note in pattern is kept in `uiNote` (from 0-MAX_NOTES - 1), and current voice you are editing in `uiVoice` (from 0-MAX_VOICES - 1). Current pattern for each voice is stored in `seqVoicePattern[<voice>]`.

Ideas for improvement

- 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.
- Add potentiometers to control LCD contrast and LED (backlight) strength.