The idea is to create a software synth using the arduino or some similar microcontrollerboard.

It has a number of virtual modules in the user interface, each with a potentiometer for adjusting a parameter, and some visualization of the current module mode, the selected parameter, and the current parameter value. Visualization could be done with leds and laser printed transparencies if the parameters and modes are more or less fixed (the transparencies can always be upgraded), or some kinds of screens.

Reference material:
https://en.wikipedia.org/wiki/Modular_synthesizer

Module types

- Oscilator
    - Parameters
        - Wave type
            - Sine
            - Triangle
            - Sawtooth forward
            - Sawtooth backward
            - Square
        - Frequency
            - Can range from very low frequencies (0.01 Hz or so) to audio frequencies (10+khz).
        - Amplitude
        - Randomness / jitter
            - Randomness is band limited to the selected frequency, and follows the wave type.
            - TODO: What parameter to use for adding non-band limited noise, or different noise types? Maybe we'll need some noise type parameter / mode as well?
        - Pulse width?
        - Offset?

Module interface

Each module could have one rotary encoder (with press button), a 2x8 character LCD screen (around 2+€ on ebay when bought in bulk), and maybe some rgb illumination (ideally a modified LCD backlight) to visualize e.g. module type. Using a LCD allows easy addition of more module types later, and also allows the system to be used for other purposes.

The flow of data between modules could be visualized with led illuminated traces in the control panel, e.g. when a control is manipulated its input and output are shown, and it could be possible to change (or mix in) inputs by pressing / turning the knobs for the other modulators.

Mozzi + LCD + Knob
Problem: LCD library uses delays / timing, mozzi turns them off.

```
#include <MozziGuts.h>
#include <Oscil.h>
#include <tables/cos2048_int8.h> // table for Oscils to play
#include <utils.h>
#include <fixedMath.h>
#include <EventDelay.h>
#include <Smooth.h>

#include <LiquidCrystal.h>

#define CONTROL_RATE 256 // powers of 2 please



int buttonPin = A0;
int knobA = A1;
int knobB = A2;
int lastAPos = LOW;
int knobPos = 0;

long lastReadTime = 0;
long counter = 0;

boolean oldButtonState = false;

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);


Oscil<COS2048_NUM_CELLS, AUDIO_RATE> aCarrier(COS2048_DATA);
Oscil<COS2048_NUM_CELLS, AUDIO_RATE> aModulator(COS2048_DATA);
Oscil<COS2048_NUM_CELLS, CONTROL_RATE> kModIndex(COS2048_DATA);

// The ratio of deviation to modulation frequency is called the "index of modulation". ( I = d / Fm )
// It will vary according to the frequency that is modulating the carrier and the amount of deviation.
// so deviation d = I * Fm
// haven't quite worked this out properly yet...

Q8n8 mod_index;// = float_to_Q8n8(2.0f); // constant version
Q16n16 deviation;

Q16n16 carrier_freq, mod_freq;

// FM ratio between oscillator frequencies, stays the same through note range
Q8n8 mod_to_carrier_ratio = float_to_Q8n8(3.f);
```

```
EventDelay kNoteChangeDelay(CONTROL_RATE);

// for note changes
Q7n8 target_note, note0, note1, note_upper_limit, note_lower_limit, note_change_step, smoothed_note;

// using Smooth on midi notes rather than frequency,
// because fractional frequencies need larger types than Smooth can handle
// Inefficient, but...until there is a better Smooth....
Smooth <int> kSmoothNote(0.95f);

void setup(){
  kNoteChangeDelay.set(768*2); // ms countdown, taylored to resolution of CONTROL_RATE
  kModIndex.setFreq(.768f); // sync with kNoteChangeDelay
  target_note = note0;
  note_change_step = Q7n0_to_Q7n8(3);
  note_upper_limit = Q7n0_to_Q7n8(50);
  note_lower_limit = Q7n0_to_Q7n8(16);
  note0 = note_lower_limit;
  note1 = note_lower_limit + Q7n0_to_Q7n8(5);
  startMozzi(CONTROL_RATE);

  setupUi();

}

void setFreqs(Q8n8 midi_note){
  carrier_freq = Q16n16_mtof(Q8n8_to_Q16n16(midi_note)); // convert midi note to fractional frequency
  mod_freq = ((carrier_freq>>8) * mod_to_carrier_ratio)  ; // (Q16n16>>8) * Q8n8 = Q16n16, beware of
overflow
  deviation = ((mod_freq>>16) * mod_index); // (Q16n16>>16) * Q8n8 = Q24n8, beware of overflow
  aCarrier.setFreq_Q16n16(carrier_freq);
  aModulator.setFreq_Q16n16(mod_freq);
}

void updateControl(){

  checkUiInput();

  // change note
  if(kNoteChangeDelay.ready()){
    if (target_note==note0){
      note1 += note_change_step;
      target_note=note1;
    }
    else{
      note0 += note_change_step;
      target_note=note0;
    }

    // change direction
    if(note0>note_upper_limit) note_change_step = Q7n0_to_Q7n8(-3);
    if(note0<note_lower_limit) note_change_step = Q7n0_to_Q7n8(3);
```

```
    // reset eventdelay
    kNoteChangeDelay.start();
  }

  // vary the modulation index
  mod_index = (Q8n8)350+kModIndex.next() + knobPos * 50;

  // here's where the smoothing happens
  smoothed_note = kSmoothNote.next(target_note);
  setFreqs(smoothed_note);

  counter++;
}


int updateAudio(){
  Q15n16 modulation = deviation * aModulator.next() >> 8;
  return (int)aCarrier.phMod(modulation);
}


void loop(){
  audioHook();
}




void setupUi() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);

  // Print a message to the LCD.
  lcd.print("Hello, World!");

  pinMode(buttonPin, INPUT);
  digitalWrite(buttonPin, HIGH);

  pinMode(knobA, INPUT);
  digitalWrite(knobA, HIGH);
  pinMode(knobB, INPUT);
  digitalWrite(knobB, HIGH);
}

void checkUiInput() {
  boolean changes = false;

  boolean state = digitalRead(buttonPin);
  if (oldButtonState != state) {
    onKnobPressChange(state);
```

```
    oldButtonState = state;
    changes = true;
  }


  long now = counter;
  if (lastReadTime + 2 >= now || now < lastReadTime + 100 ) {
    lastReadTime = now;

    int aPos = digitalRead(knobA);
    if ((lastAPos == LOW) && (aPos == HIGH)) {
      int delta = 0;
      if (digitalRead(knobB) == LOW) {
        delta = -1;
      } else {
        delta = 1;
      }

      knobPos += delta;

      onKnobTurn(delta, knobPos);
      changes = true;
    }
    lastAPos = aPos;
  }

  if (changes) {
    doUiOutput();
  }
}


void onKnobPressChange(boolean knobPressState) {
    // Do state change stuff
}

void onKnobTurn(int delta, int knobPos) {
      // TODO : Handle knob change here
}


void doUiOutput() {
  lcd.setCursor(0, 1);
  lcd.print(knobPos);
  lcd.print(" ");
  if (oldButtonState) lcd.print("BUTTON UP");
  else lcd.print("BUTTON DOWN");

}
```

VErsion 2:

```
/*  Example of simple FM with the phase modulation technique,
 *  using Mozzi sonification library.
 *
 *  Demonstrates Oscil::phMod() for phase modulation,
 *  Smooth() for smoothing control signals,
 *  and Mozzi's fixed point number types for fractional frequencies.
 *
 *  Also shows the limitations of Mozzi's 16384Hz Sample rate,
 *  as aliasing audibly intrudes as the sound gets brighter around
 *  midi note 48.
 *
 *  Circuit: Audio output on digital pin 9.
 *
 *  Mozzi help/discussion/announcements:
 *  https://groups.google.com/forum/#!forum/mozzi-users
 *
 *  Tim Barrass 2012.
 *  This example code is in the public domain.
 */

#include <tables/cos8192_int8.h> // table for Oscils to play

#include <MozziGuts.h>
#include <Oscil.h>
#include <tables/cos2048_int8.h> // table for Oscils to play
#include <utils.h>
#include <fixedMath.h>
#include <EventDelay.h>
#include <Smooth.h>

#include <LiquidCrystal.h>

#define CONTROL_RATE 64 // powers of 2 please
```

```
int buttonPin = A0;
int knobA = A1;
int knobB = A2;
int lastAPos = LOW;
int knobPos = 0;

int parameter = 0;
int parameterCount = 3;
int parameterValues[] = {100, 10, 20};
int parameterMin[] = {0, 0, 0};
int parameterMax[] = {1000, 200, 30};
String parameterNames[] = {"Intensity", "Wave Freq", "Vibrato Freq"};

long lastReadTime = 0;
long counter = 0;

boolean oldButtonState = false;

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);


Oscil<COS8192_NUM_CELLS, AUDIO_RATE> sineWave(COS8192_DATA);
Oscil<COS8192_NUM_CELLS, AUDIO_RATE> vibrato(COS8192_DATA);

long intensity = 300;


void setup(){
  startMozzi(CONTROL_RATE);
  sineWave.setFreq(mtof(84.f));
  vibrato.setFreq(15.f);

  setupUi();

}

void updateControl(){

  checkUiInput();

  intensity = parameterValues[0];
  sineWave.setFreq(mtof(1.0f * parameterValues[1]));
  vibrato.setFreq(1.0f * parameterValues[2]);

}


int updateAudio(){
    long vibratoVal = intensity * vibrato.next();
    return (int)sineWave.phMod(vibratoVal);
}
```

```
void loop(){
  audioHook();
  counter++;
  checkUiInput();
}




void setupUi() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);

  // Print a message to the LCD.
  lcd.print("Hello, World!");

  pinMode(buttonPin, INPUT);
  digitalWrite(buttonPin, HIGH);

  pinMode(knobA, INPUT);
  digitalWrite(knobA, HIGH);
  pinMode(knobB, INPUT);
  digitalWrite(knobB, HIGH);
}

void checkUiInput() {
  boolean changes = false;

  boolean state = digitalRead(buttonPin);
  if (oldButtonState != state) {
    onKnobPressChange(state);
    oldButtonState = state;
    changes = true;
  }


  long now = counter;
  if (lastReadTime + 10000L >= now) {
    lastReadTime = now;

    int aPos = digitalRead(knobA);
    if ((lastAPos == LOW) && (aPos == HIGH)) {
      int delta = 0;
      if (digitalRead(knobB) == LOW) {
        delta = -1;
      } else {
        delta = 1;
      }
```

```
      knobPos += delta;

      onKnobTurn(delta, knobPos);
      changes = true;
    }
    lastAPos = aPos;
  }

  if (changes) {
    doUiOutput();
  }
}


void onKnobPressChange(boolean knobPressState) {
    // Do state change stuff
    if (knobPressState) {
      parameter++;
      if (parameter < 0) parameter += parameterCount;
      parameter %= parameterCount;
    }
}

void onKnobTurn(int delta, int knobPos) {
      // TODO : Handle knob change here

  if (!oldButtonState) {
    parameter += delta;
    if (parameter < 0) parameter += parameterCount;
    parameter %= parameterCount;
  }
  else {
    parameterValues[parameter] += delta;

    if (parameterValues[parameter] < parameterMin[parameter]) parameterValues[parameter] =
parameterMin[parameter];
    if (parameterValues[parameter] > parameterMax[parameter]) parameterValues[parameter] =
parameterMax[parameter];
  }

}


void doUiOutput() {

  lcd.setCursor(0, 0);
  lcd.print(parameterNames[parameter]);
  lcd.print("               ");

  lcd.setCursor(0, 1);
  lcd.print(parameterValues[parameter]);
  lcd.print("               ");
}
```