MCO: MIDI Controlled Oscillator

By jean-claude.feltes@education.lu

The idea

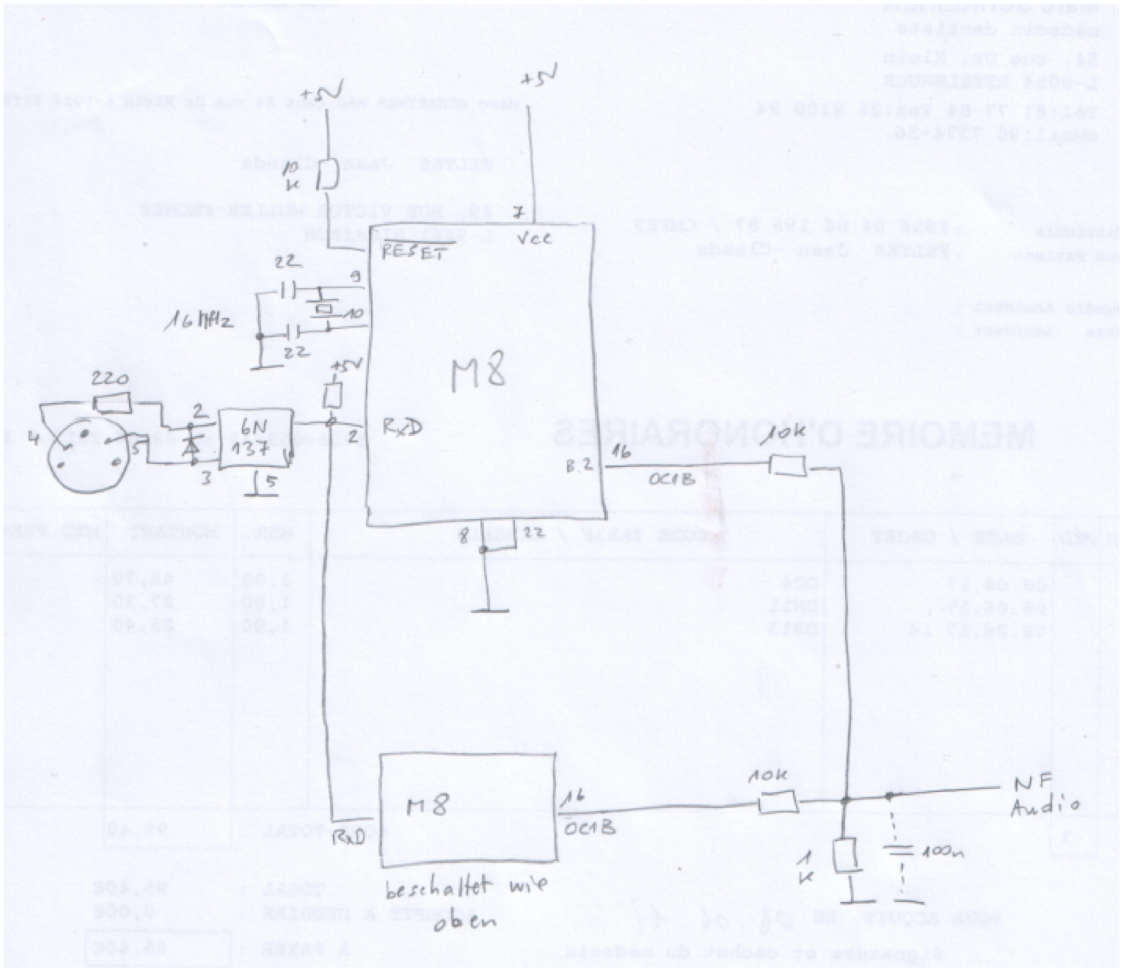
Around 1975 I was fascinated by electronic music. It was the time Emerson, Lake and Palmer, Pink Floyd and others were using the Moog synthesizer. Would it be possible for an amateur to build such an instrument? Much of my time was spent on the answer to this question.

Later fantastic keyboards were (too) cheap and the interest of selfmade instruments was vanishing.

But today there is an interesting come back, as there are new possibilities using software defined instruments on computers ore microcontrollers.

Maybe there could be a revival of my old interests and maybe some of my old circuits could be recycled?

A fist test with one or two Mega8



The heart of a Moog synthesizer was the VCO, a voltage controlled oscillator.
Nowadays we would use a MIDI keyboard and so need an MCO, a MIDI controlled oscillator.

In the Arduino MIDI library I found the SimpleSynth in the Examples section:
File - Examples – MIDI – SimpleSynth

This gives a monophonic MCO playing always the highest note, if more than one note is pressed.

The output is a square wave, sounding not too good, but less bad than expected, except for a clicking noise when switching from silence to note.

From my experiments with analog synthesizers I remember that there is a remarkable better sound when several (2 or 3) oscillators are coupled to play the same note.
If we detune the oscillators by a faint amount, we have a small beat and the sound gets by far more interesting.

So I used two Mega8 as oscillators coupled the way the schematic shows.
I expected some beat between the 2 added signals, due to tolerances in the crystal frequency.
But no, there was none! The crystals must be very precise in frequency.

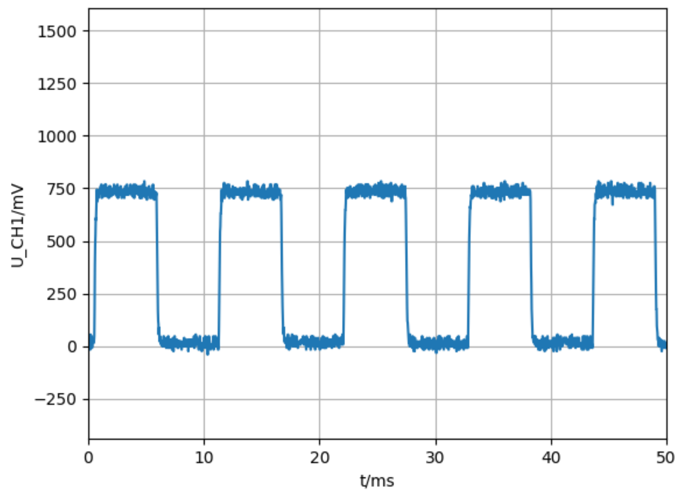
Next variation, add a detune factor in the handleNotesChanged function.

```
void createTone(float frequency){
  tone(sAudioOutPin, frequency);
}

void switchOffTone(){
  noTone(sAudioOutPin);
}

//-----------------------------------------------------------------------------

void handleNotesChanged(bool isFirstNote = false)
{
  if (midiNotes.empty())
  {
    handleGateChanged(false);
    switchOffTone();     // Remove to keep oscillator running during envelope
release.
  }
  else
  {
    // Possible playing modes:
    // Mono Low:  use midiNotes.getLow
    // Mono High: use midiNotes.getHigh
    // Mono Last: use midiNotes.getLast

    byte currentNote = 0;
    if (midiNotes.getLast(currentNote))
    {
      //CHANGED
      float f = sNotePitches[currentNote];
      //f = 0.995 * f;
      f = detuneFactor * f;
      createTone(f);
      //END CHANGED


      if (isFirstNote)
      {
        handleGateChanged(true);
      }
      else
      {
        pulseGate(); // Retrigger envelopes. Remove for legato effect.
      }
    }
  }
}
```

I also put the sound creating functions extra, to more easily try other sound producing libraries.

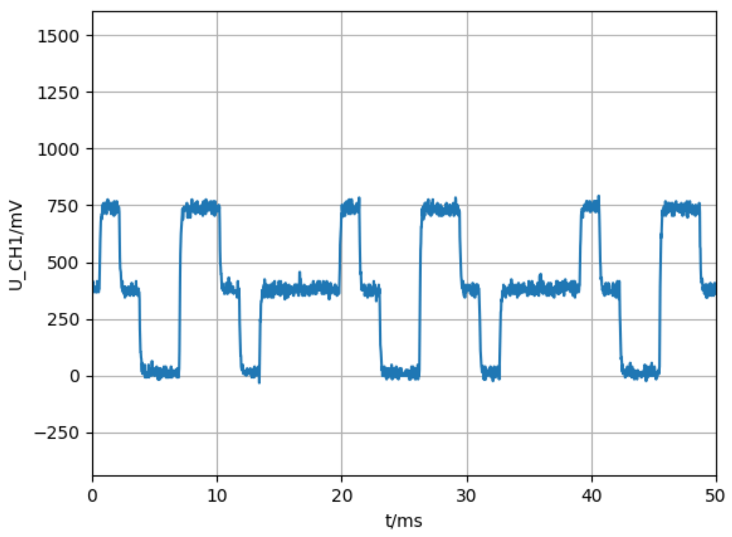
Naturally the detuneFactor has to be declared as global variable in the beginning of the code:
`float detuneFactor = 1.0;`
Now some interesting experiments are possible.
One of the VCO stays with detuneFactor = 1.0, the other is set to different values.

Detune = 1.0



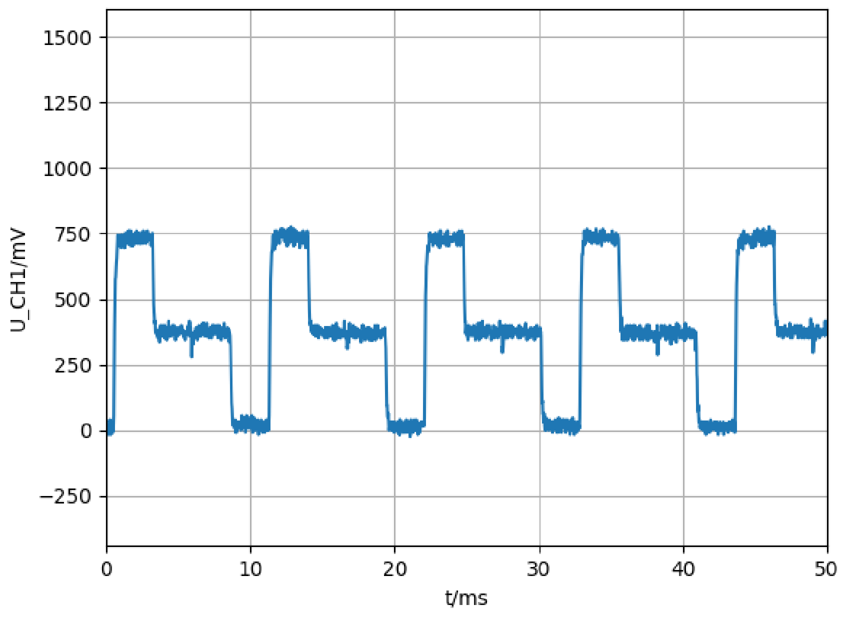
The sound is that of a typical rectangle signal, the absence of even harmonics makes it somewhat “hollow”.

Detune = 1.5



As there is a fifth (Quinte) added, the sound has “power”.

Detune = 2.0



As we have a added a signal of double frequency (one octave higher) we have rich even harmonics that sound a bit like a sawtooth signal (there are even more harmonics than in a sawtooth signal).

This would be a good starting point if we want to filter the signal.

The sound gets more interesting if we set the detune factor not to the exact values 1, 1.5, 2 but to slightly different ones like 1.02, 1.52, 2.02. In this case we have a beat in the signal, the sound is richer, as if two instruments were playing the same note.

The program can be modified so that on a button press from one of three buttons we have different detune factors:

```
void readDetuneFactor(){

 if (digitalRead(button1) == 0){
   detuneFactor = 1.02;
   }
 if (digitalRead(button2) == 0){
   detuneFactor = 2.02;
 }
  if (digitalRead(button3) == 0){
   detuneFactor = 1.52;
   }

}
//----------------------------------------

void loop(){
  MIDI.read();
  readDetuneFactor();

}
```

Another idea would be to make the detune factor variable by adding a potentiometer.
This has 2 reasons not to work as desired:

1. Once a note is started, the potentiometer value would have no effect. So it couldn't be used to bend the pitch.
2. By adding this, we go beyond the program memory of a Mega8.

Using a Mega32 (or Mega16)

I could not resist the idea to try the variation of the detune factor with a potentiometer. This was easy as I had some Mega32 boards from earlier projects.

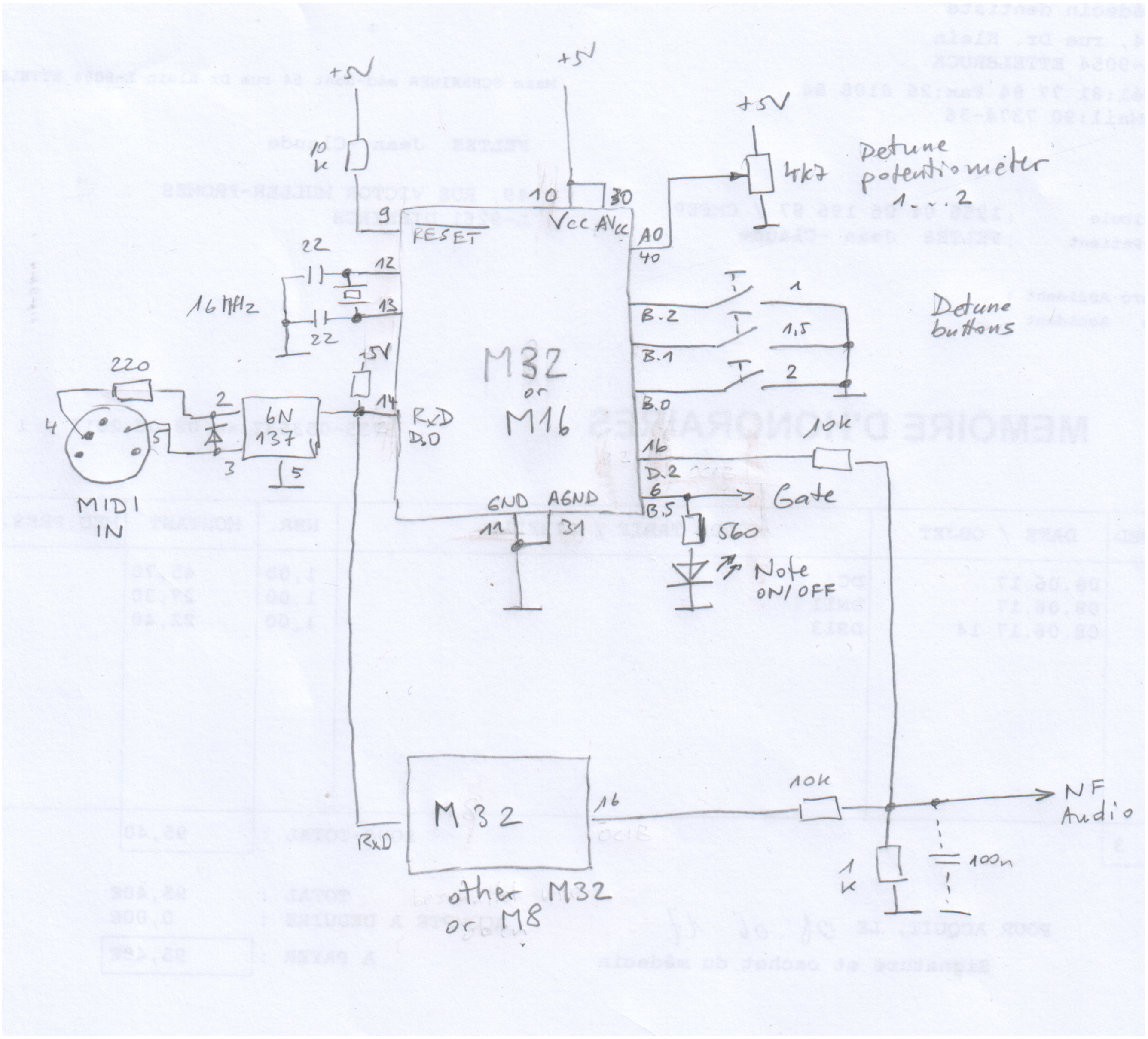
With this now the variation with a potentiometer was easily done, and it worked!

```
void readDetuneFactor(){

  int potValue = analogRead(analogInPin);

 float detuneFactor2 = potValue/1023.0 + 1;


 if (digitalRead(button1) == 0){

   detuneFactor1 = 1.0 ;

   }
 if (digitalRead(button2) == 0){
   detuneFactor1 = 2.0 ;
 }
 if (digitalRead(button3) == 0){
   detuneFactor1 = 1.5;
   }
 detuneFactor = detuneFactor1 * detuneFactor2;
}
```

By the way, the audio output pin can be any pin and is not restricted to the timer PWM outputs.

Maybe a 4th button that gives us a fourth interval (Quarte) could be interesting.

MCO with Mega32



Code

http://staff.ltam.lu/feljc/electronics/synthesizer/MCO_Mega32_VCO.zip