Arduino generates itself wave tables for the Open.Theremin UNO project

1. Introduction

The Open.Theremin UNO project consists of an Arduino UNO processor board with an additional theremin circuit board (« shield » in Arduino language) and corresponding software which takes the pulse widths coming from the heterodyne circuits on the shield, and processes them to obtain 2 integer values, one for pitch and one for volume. The pitch value is used to step through a wave form lookup table which is defined as an array of 1024 integers in a separate file. The resulting y(t) value is then multiplied by the volume variable.

The Arduino UNO has not the computing power to generate a complex waveform in real time, thus the external lookup table is the only way to obtain nice waveforms. But most people who build and play the Open.Theremin neither have the programming and mathematical skills nor the patience to calculate and to write new wavetable files for their instrument.

So, why not use the Arduino to calculate different wave tables offline in a different sketch and then use the result by recompiling and uploading the Open.Theremin software again?

2. How it works

Since the Arduino's only way to communicate with the development environment (IDE) is the serial port, the only way is to make the Arduino calculate a wave table and to output it in the serial monitor window which can be opened from the IDE's Tools menu. Once the Arduino will have finished writing the new wave table, the user will just have to select the whole content of the serial window with Ctrl-A, and copy t with Ctrl-C. Then he'll have to open the Open.Theremin sketch, select the « theremin_sintable.c » file tab, delete everything which is in its code window, and paste the new wave table code here. Recompiling and uploading the Open.Theremin software will make the Open.Theremin play with the new sound.

3. How the waveform is generated in theremins

The algorithm which calculates the new waveform emulates the wave shaping circuit of most recent Moog Theremins. It is basically similar to the circuits used in the Ethervox, Series 91, Etherwave Standard/Plus, and Etherwave Pro theremins. The final timbre difference between these instruments comes mostly from using different presets, slightly different input signal waveforms (from the heterodyning mixer) and from the more or less sophisticated post wave shaping filter circuits. The filtering can not be taken into account when generating simple wave tables since these force us using the same waveform for whatever audio frequency, thus we'll have to rely on the Open.Theremin's post DAC low pass filter. I decided to start from a triangle signal as in the Series 91, Ethervox, and Etherwave Pro theremins, while the Etherwave Standard/Plus use the abs(cos(x/2)) signal from the diode demodulator which is slightly different.

What is common to all the Moog instruments cited above is that they use a LM13700 OTA in the wave shaping circuit. Its differential input stage will always be overdriven by the demodulator signal to obtain some clipping which adds harmonics to the signal. The degree of clipping (harder or softer) can be adjusted by an additional diode bias current through both inputs. This corresponds to the « Brightness » potentiometer which decreases the bias current when turned clockwise: ***More brightness => less bias current => harder clipping***.

A second way to influence the waveform is its symmetry before clipping occurs. This is done by adding an offset voltage to the input by the means of the « Waveform » potentiometer. Without any offset, the signal is symmetrical. With increasing offset, the signal becomes more and more asymmetric, leading again to more harmonics. ***More waveform => more offset =>less symmetry***.

Both settings influence one another. That means that if there is more brightness, variations of the waveform parameter will be better audible and vice versa. The resulting waveforms can be seen in the graphic table at the end of this article.

4. How that can be translated into code

It's all about mathematics. First we have to scale the time axis, so that 1024 samples will give a full wave period:
t(x) = 2 * pi * x / 1024 or simplified t(x) = x * pi / 512 (1)

Now, let's build a normalized triangle wave from that:
y1(t) = arcsin(sin(t)) (2)

Take an offset value wf (waveform) between 0 and 255, scale it down and add it:
y2(t) = y1(t) + 0.8 * wf / 255 (3)

Take a bias value br (brightness) between 0 and 255, scale it down and multiply it do drive the signal more or less into saturation:
y3(t) = y2(t) * 6 * (1 + 3 * br / 255) / pi (4)

Apply the clipping function of a differential transistor pair and scale the result to the desired amplitude:
y4(t) = 2048 * tanh(y3(t)) (5)

Since y3(t) gives values from -1 to 1, the upscaled y4(t) will give values from -2048 to 2048. The 12bit DAC will only accept values from -2048 to 2047, thus we'll have to squeeze the output signal slightly:
y5(t) = constrain(y4(t), -2048, 2047) (6)

Substituting and assembling all these equations and doing a loop for the whole range gives:

```
for( int x = 0; x < 1024; x++) {
    y = constrain( 2048 * tanh( ( arcsin( sin( x * pi / 512) ) + 0.8 * wf / 255 ) * 6 * (1 + 3 * br / 255 ) /
        pi ), -2048, 2047);
}
```

That's mostly it. In reality, we'll use some #defines to make it easier to change the table width and amplitude so that the code could be used for different hardware configurations, and we will use some pseudo const intermediate variables to make the code better readable and to ease its compiling.

Although the Arduino UNO is rather asthmatic when it comes to floating point maths, I decided to write the code using float variables and constants to allow precise scaling for whatever wave table width and amplitude. The time for generating a wave table remains under one minute, though.

5. The code

Everything is done in the setup() routine, the loop will turn empty when the wave table is generated:

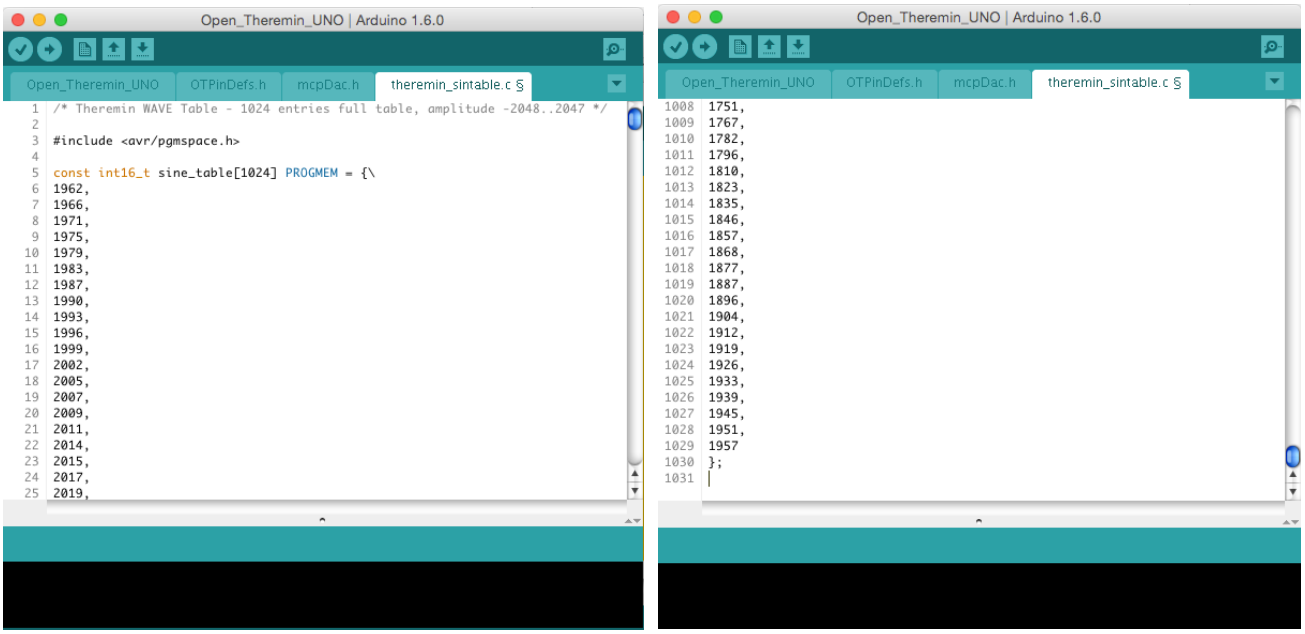
```
/* Generates the (alternative) content of a theremin_sintable.c file
   for the open.theremin.uno project. Compile and upload this sketch
   to your Arduino and open immediately the serial monitor where the
   content of the future new theremin_sintable.c file will appear.
   Once it will be done, click somewhere in the serial monitor window,
   then press Ctrl-A to select all, and Ctrl-C to copy it. Then move
   to your second Arduino IDE window and paste everything into an empty
   theremin_sintable.c file and you are done */

/* Just a simple idea by Theremingenieur Thierry Frenkel 2015 */

/* Everybody is free to use and to modify this code for non-commercial
   purposes as long as you mention my name and praise my genius! ;-) */

#define tableWidth 1024
#define dacResolutionBits 12

void setup() {
  Serial.begin(57600);
  // Allow the user some time to open the serial monitor
  delay(5000);
  // Print the file header
  Serial.print("/* Theremin WAVE Table - 1024 entries full table, amplitude -2048..2047 */\n\n");
  Serial.print("#include <avr/pgmspace.h>\n\n");
  Serial.print("const int16_t sine_table[1024] PROGMEM = {\\ \n");

  // here the parameters you can play with:
  uint8_t  brightness = 128; //you might put values from 0 to 255
  uint8_t  waveform = 128; //you might put values from 0 to 255

  // some preliminary parameter conditioning, don't touch!
  const float    pi = 3.1415926535;
  float    br = (1.0f + 3.0f * (float)(constrain(brightness, 0, 255) / 255.0f)) * 6.0f / pi;
  float    wf = 0.8f * (float)(constrain(waveform, 0, 255) / 255.0f);
  int16_t  ampl = 1 << (dacResolutionBits - 1);
  int16_t  dacMax = ampl - 1;
  int16_t  dacMin = 0 - ampl;
  float    hp = pi * 2.0f / tableWidth;
  // we need values for intermediate and final results in the function's equation
  int16_t y;

  // now the loop to generate the time steps and function values, and output the latter
  for (int16_t t = 0; t < tableWidth; t++) {
    y = constrain(ampl * tanh((asin(sin(t * hp)) + wf) * br), dacMin, dacMax);
    // formatted output, add a comma after each line but the last
    Serial.print(y, DEC);
    if (t < (tableWidth - 1)) {
      Serial.print(",\n");
    } else {
      Serial.print("\n");
    }

  }
  Serial.print("};\n");
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

6. Example output

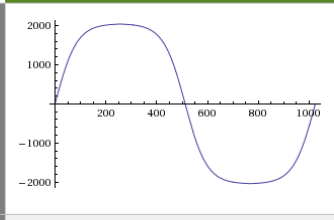
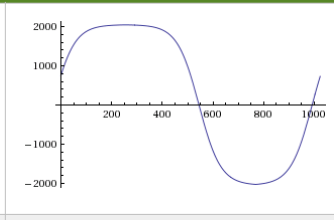
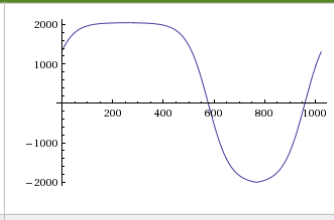
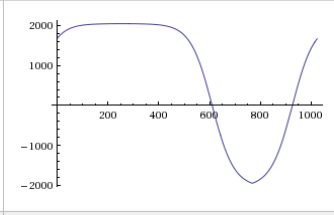
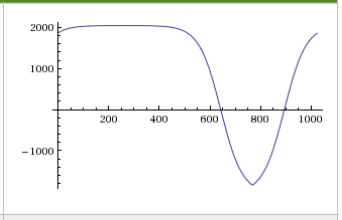
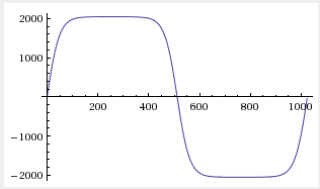
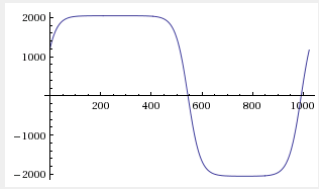
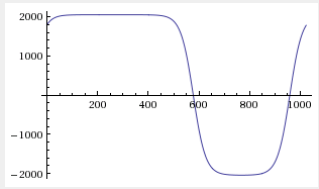
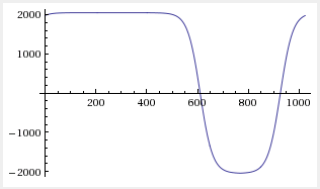
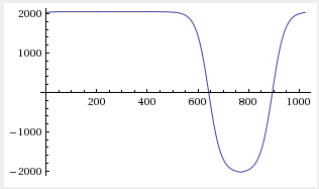
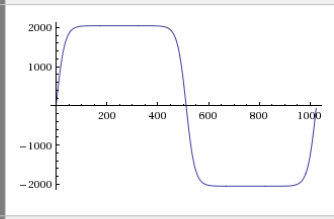
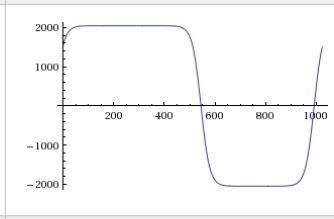
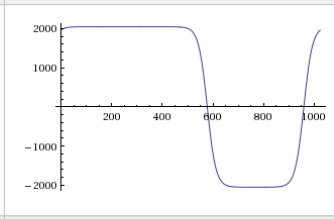
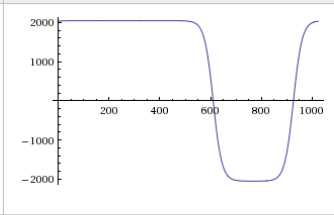
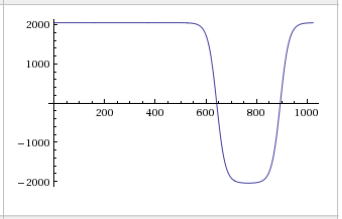
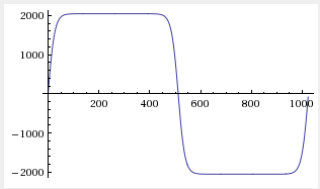
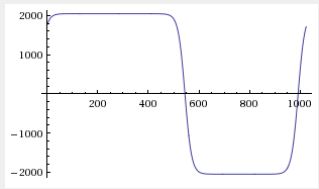
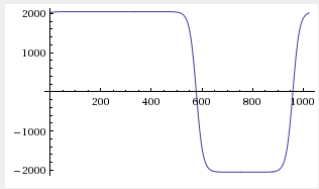
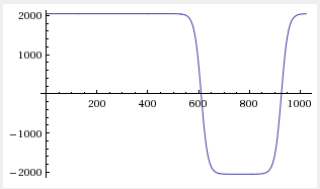
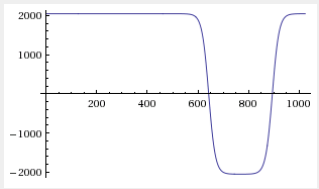
What you will find in the serial monitor window looks like that:

```
/* Theremin WAVE Table - 1024 entries full table, amplitude -2048..2047 */

#include <avr/pgmspace.h>

const int16_t sine_table[1024] PROGMEM = {\
1962,
1966,
1971,
.
.
.
1945,
1951,
1957
};
```

After pasting everything into the theremin_sintable.c file it should look like that, the first numeric value in line 6 and the last one in line 1029 with the closing bracket in line 1030:



Now, have fun!

Brightness \ Waveform	0	64	128	192	255
0					
64					
128					
192					
255