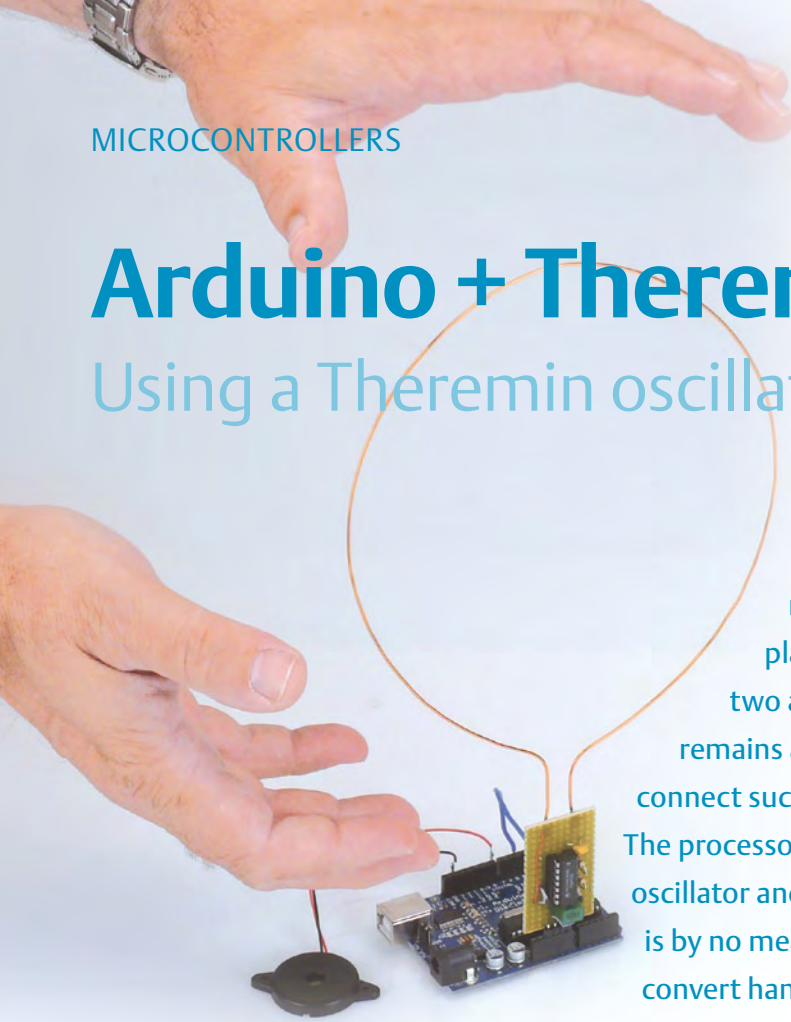
Arduino + Theremin = Theremino

Using a Theremin oscillator as a proximity sensor

By Martin Nawrath (Academy of Media Arts, Cologne, Germany)



The Theremin is one of the very first electronic musical instruments, dating back to the 1920s. It is played by the musician bringing his hands close to its two antennas. The oscillator at the heart of the Theremin remains an interesting circuit in itself, and in this project we connect such an oscillator to an Arduino microcontroller board. The processor can detect the shifts in the HF signal from the oscillator and convert them into an audible sound. However, this is by no means the only possibility: we can also use the circuit to convert hand and body movements into control signals for other musical instruments, servos and computers.

This tiny oscillator circuit, when coupled with the software running on the Arduino microcontroller board, has a huge range of potential applications, allowing proximity-based control of any other circuit or system.

The circuit has already found use in many installations and objects at the author's college. One example is where the device is interfaced to the Max/MSP music and multimedia development environment, which is capable of producing sounds that provide a pleasant contrast to the square waves that the Arduino can generate directly.

What makes a theremin a theremin?

The *Elektor* editorial team felt it was possible that calling this project a 'Theremin' might be a little misleading. The musical instrument named after Russian inventor and engineer Léon Theremin (born Lev Termen) consists of two antennas, each connected to an oscillator. One oscillator controls the pitch of the instrument and the other the volume. The original Theremin is a purely analogue device, with the conversion from the modulated high-frequency signal to an audible frequency being performed by a superhet circuit: the oscillator output is mixed with a fixed frequency generated by a further oscillator, arranged so that the difference frequency is in the audible range.

Our circuit has just one oscillator, nevertheless designed around the same principle as the Theremin. The oscillator is connected to an antenna, and the oscillator's frequency changes when a human hand (or other electrically conductive object) is brought near to the antenna. The additional capacitance of the hand affects the frequency of the resonant circuit in the oscillator (**Figure 1**). It seems reasonable to call this circuit a 'Theremin oscillator'. As in the original Theremin, we convert the modulated high-frequency signal into the audible range; but rather than using a superhet converter, we do the job in software on the Arduino board. Our 'Theremino' therefore replicates just the pitch control part of the original instrument. It would be possible to build a second Theremin, suitably modified to provide a volume control, to emulate the analogue Theremin fully.

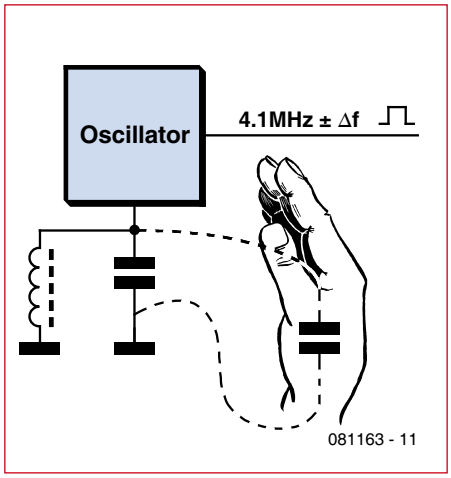


Figure 1. The parallel capacitance of a human hand near to the oscillator affects its frequency.

LC oscillator using a 74HC00

The circuit shown in **Figure 2** consists of an amplifier, made to oscillate by coupling its output signal back to its input. At the input to the amplifier is a parallel resonant circuit, made from a coil and a capacitor, which determines the frequency of the oscillator. Any extra parallel capacitance due to the connected antenna will affect this frequency.

The amplifier is made from two NAND gates from a 74HC00 connected in series. Each is equipped with a resistor (R1 and R2) to provide negative feedback, which causes the gates to behave as amplifiers. C3 provides feedback from the output to the resonant circuit at the input consisting of L1 and C1. C4 couples the signal on the resonant circuit into the input of the amplifier. The resonant circuit is also connected to the antenna. The theoretical resonant frequency of the LC combination is 4.11 MHz.

The oscillator is followed by the two remaining gates of the 74HC00, used to square up its output waveform into a TTL-compatible signal. This signal is suitable for connection to a digital input on the Arduino board.

Components and construction

Capacitor C1 in the LC network should be a ceramic NP0 type (such as Farnell order code 9411720) for best temperature stability. L1 should have a high Q factor. A suitable type is the Fastron SMCC-100K-02, which has a Q factor of 65; it is difficult to do better than that, even with a hand-wound coil.

The circuit can be built, like the author's prototype, on a small piece of prototyping board. To reduce drift with temperature, it is a good idea to mount the board in a small plastic enclosure. The software running on the Arduino board also has features to help compensate for temperature drift and component tolerances.

The antenna that is connected to the LC circuit should be no longer than about one metre (3 feet). A loop of 1.5 mm copper wire (or, for greater mechanical strength, steel wire) works well.

With the antenna operating at 4 MHz metal objects and cables near to it (including USB cables) will also act as parasitic antennas. It is therefore important to keep the unit fixed in place to avoid unwanted frequency drift.

Arduino software

For our tests in the Elektor lab we used an Arduino Diecimila board [1] with software downloaded from the author's project website. In principle any Arduino board could be used, including the 'Elektorino' design that we published in February 2009 [2]. The oscillator circuit of Figure 2 is provided with +5 V and ground from the Arduino board and the oscillator output (f_{out}) is connected to digital pin 5 (PD5, pin 11 on the ATmega168 [3]). This pin has the extra function of acting as an input to hardware counter/timer Timer1 in the ATmega168. To detect the frequency shifts of the oscillator, an accurate fre-

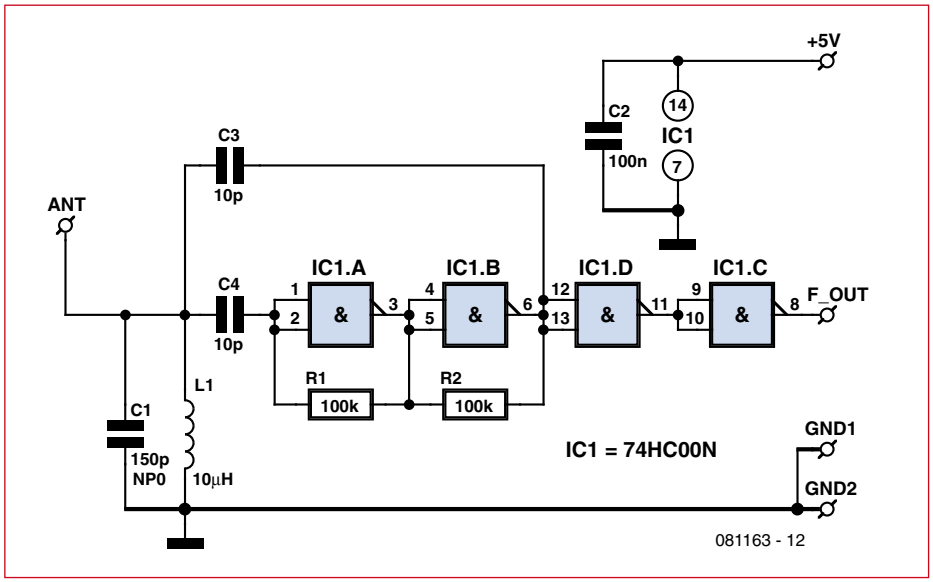


Figure 2. Circuit of the LC oscillator using the 74HC00. To increase the sensitivity of the circuit to the proximity of a human hand, an antenna, made from a length of wire, is connected to the LC network.

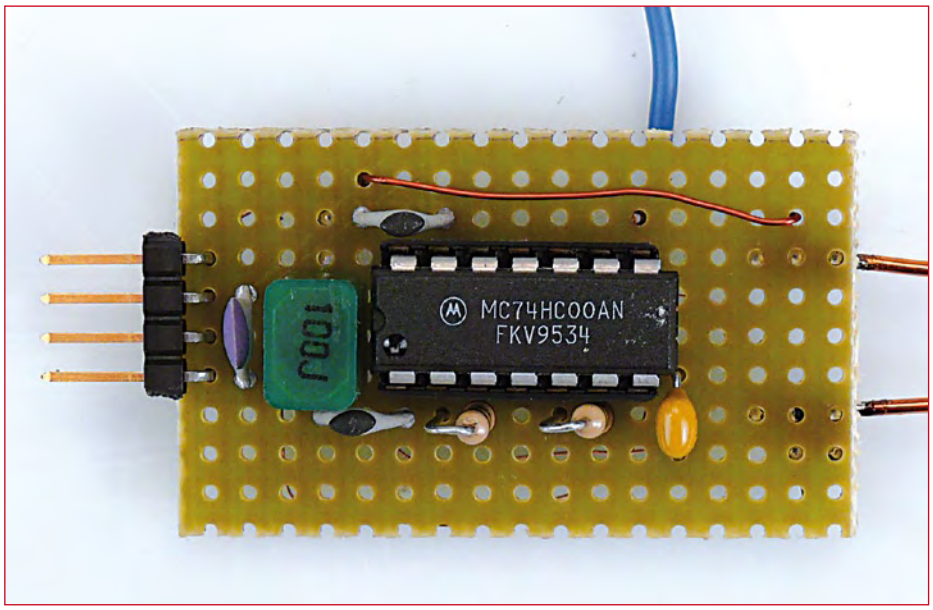


Figure 3. The oscillator can be built on a small piece of prototyping board.

Listing 1. Frequency is measured using Timer1 as a counter and Timer2 to measure the gate time.

```
//****************************************************************
void f_meter_start() {
  f_ready=0;                      // reset period measure flag
  i_tics=0;                        // reset interrupt counter
  sbi (GTCCR,PSRASY);              // reset prescaler counting
  TCNT2=0;                         // timer2=0
  TCNT1=0;                         // Counter1 = 0
  cbi (TIMSK0,TOIE0);              // disable Timer0 again // millis and delay
  sbi (TIMSK2,OCIE2A);             // enable Timer2 Interrupt
  TCCR1B = TCCR1B | 7;             // Counter Clock source = pin T1 , start counting now
}

//****************************************************************
// Timer2 Interrupt Service is invoked by hardware Timer2 every 2ms = 500 Hz
//  16Mhz / 256 / 125 / 500 Hz
// gate time generation for freq. measurement takes place here:

ISR(TIMER2_COMPA_vect) {

  if (i_tics==50) {          // multiple 2ms = gate time = 100 ms
                             // end of gate time, measurement ready
    TCCR1B = TCCR1B & ~7;    // Gate Off / Counter T1 stopped
    cbi (TIMSK2,OCIE2A);     // disable Timer2 Interrupt
    sbi (TIMSK0,TOIE0);      // enable Timer0 again // milli-s and delay
    f_ready=1;               // set global flag for end count period

                             // calculate now frequency value
    freq_in=0x10000 * mlt;   // multiply number of overflows by 65536
    freq_in += TCNT1;        // add counter1 value
    mlt=0;

  }
  i_tics++;                  // count number of interrupt events
  if (TIFR1 & 1) {           // if Timer/Counter 1 overflow flag
    mlt++;                   // count number of Counter1 overflows
    sbi(TIFR1,TOV1);         // clear Timer/Counter 1 overflow flag
  }

}
```

quency counter is realised in the Arduino firmware using Timer1 as a counter and Timer2 as a timebase. When suitably configured by the software, the counter increments by one for each pulse on the input pin, typically over four million times per second. Timer2 provides a gate time of exactly 1/10 s (100 ms). With an input frequency of 4.1 MHz Timer1 will increment 410 000 times during the gate period, and the frequency can be measured to a resolution of 10 Hz. This precision is required in order to detect the relatively small frequency shifts caused by the Theremin effect.

Timer1 is only a 16-bit counter and so it will overflow several times in each gate period. The overflows are counted and combined with the counter value to produce a final result at the end of the gate period. All the timing work is handled by an interrupt function, called every two milliseconds under control of Timer2 (see **Listing 1**).

At the end of the gate period a global flag variable is set. This signals to the main code (see **Listing 2**) that a new result is ready. Since we are only interested in relative changes in the input frequency and not its absolute value, we subtract the first measured frequency after power-up from each reading. If after a preset number of readings the frequency shift is less than a certain threshold value, an automatic calibration is performed: this compensates for the effect of long-term oscillator drift. The two parameters (number of readings and threshold value) can be adjusted to suit a particular application. The calculated frequency shift value (variable 'tune' in the listing) can be used as a starting point for your own application ideas.

The frequency shifts and calibration values are output on the serial port for further processing or for viewing using a terminal program on a PC.

A rudimentary DDS tone generator function is implemented in software to produce an audible frequency on port B. The signal can be heard by connecting a piezo transducer or small loudspeaker to digital pin 8 (PB0, or pin 14 of the ATmega168) via a 1 kΩ series resistor. You can see and hear the circuit in action in a video on the author's project website [4].

(081163-I)

Listing 2. Excerpt from the main loop. The variable 'tune' contains the frequency shift as produced by the proximity sensor. Automatic calibration compensates for the effect of long-term oscillator drift.

```
void loop()
{
  cnt++;

  f_meter_start();

  tune=tune+1;
  while (f_ready==0) {            // wait for period end (100ms) using interrupt
    PORTB=((dds+=tune) >> 15);    // kind of DDS tone generator: connect speaker to portb.0 = Arduino pin8
  }
 tune = freq_in-freq_zero;
 // use the tune value here for your own purposes like control of servos, midi etc.

  // startup
  if (cnt==10) {
    freq_zero=freq_in;
    freq_cal=freq_in;
    cal_max=0;
    Serial.print("** START **");
  }

  // automatic calibration
  if (cnt % 20 == 0) {   // try automatic calibration after n cycles
    Serial.print("*");
    if (cal_max <= 2) {
      freq_zero=freq_in;
      Serial.print(" calibration");
    }
    freq_cal=freq_in;
    cal_max=0;
    Serial.println("");
  }
  cal = freq_in-freq_cal;
  if ( cal < 0) cal*=-1;  // absolute value
  if (cal > cal_max) cal_max=cal;
}
```

Internet Links

[1] www.arduino.cc/en/Main/ArduinoBoardDiecimila

[2] www.atmel.com/dyn/resources/prod_documents/doc2545.pdf

[3] www.elektor.com/080931

[4] http://interface.khm.de/index.php/lab/experiments/theremin-as-a-capacitive-sensing-device/

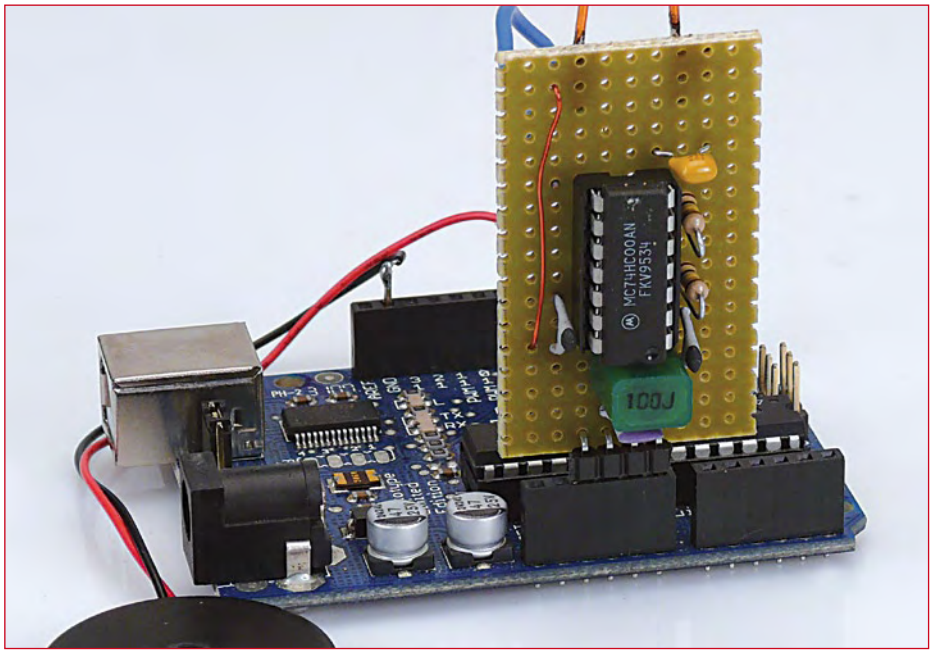


Figure 4. Arduino board with oscillator circuit and piezo transducer.