Build the ***Nano*** WAVEFORM GENERATOR

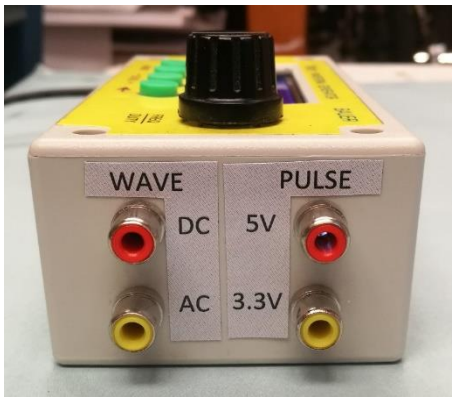
A DIY electronics project by M J Bauer

Overview

Waveform generators, also known as “function generators”, have been popular DIY projects since the beginning of hobby electronics. There are numerous designs out there offering a broad range of functional features and design complexities. The “Nano” waveform generator sits near the low end of complexity and build cost in the hardware design. However, it packs a lot of punch into an 8-bit AVR micro-controller to produce a useful instrument for testing audio and low-speed digital equipment.

The parts are housed in a small plastic box (130 x 70 x 40mm) making the unit easily portable. It is powered by a 5V USB plug-pack.

All the parts needed to build a Nano wave-generator are available at ridiculously low prices from online suppliers on AliExpress, eBay, etc. Note that delivery from Chinese suppliers, although very cheap (sometimes free) can take a while, typically 2 ~ 4 weeks.



Functional Specification

WAVE mode

Wave-shapes: SINE, TRIANGLE, SQUARE, SAWTOOTH (+ or − slope), NOISE

Frequency:

Low range: 1 Hz ~ 100 Hz in 12 preset steps

High range: 80 Hz ~ 8 kHz in 18 preset steps
or 50 Hz ~ 5 kHz continuously variable using pot

Noise: Filtered with variable cut-off frequency (first-order IIR filter) at 50, 100, 200, 400, 800, 1600, 3200 Hz, or unfiltered ("white" noise).

Output coupling: AC or DC

Output level: 100mV, 200mV, 500mV, 1V, 2V, 4V peak-to-peak

PWM DAC resolution: 8 bits (0.4% FS, approx. 4mV at 1Vp-p)

PULSE mode

Frequency:

Low range: 1 Hz ~ 1000 Hz in 16 preset steps

High range: 1 kHz ~ 4 MHz in 16 preset steps

Duty / pulse width: 1 ~ 99 % variable using pot, quantized values

Output level: 3.3V or 5V fixed, 20mA source/sink (at 5V)

Design

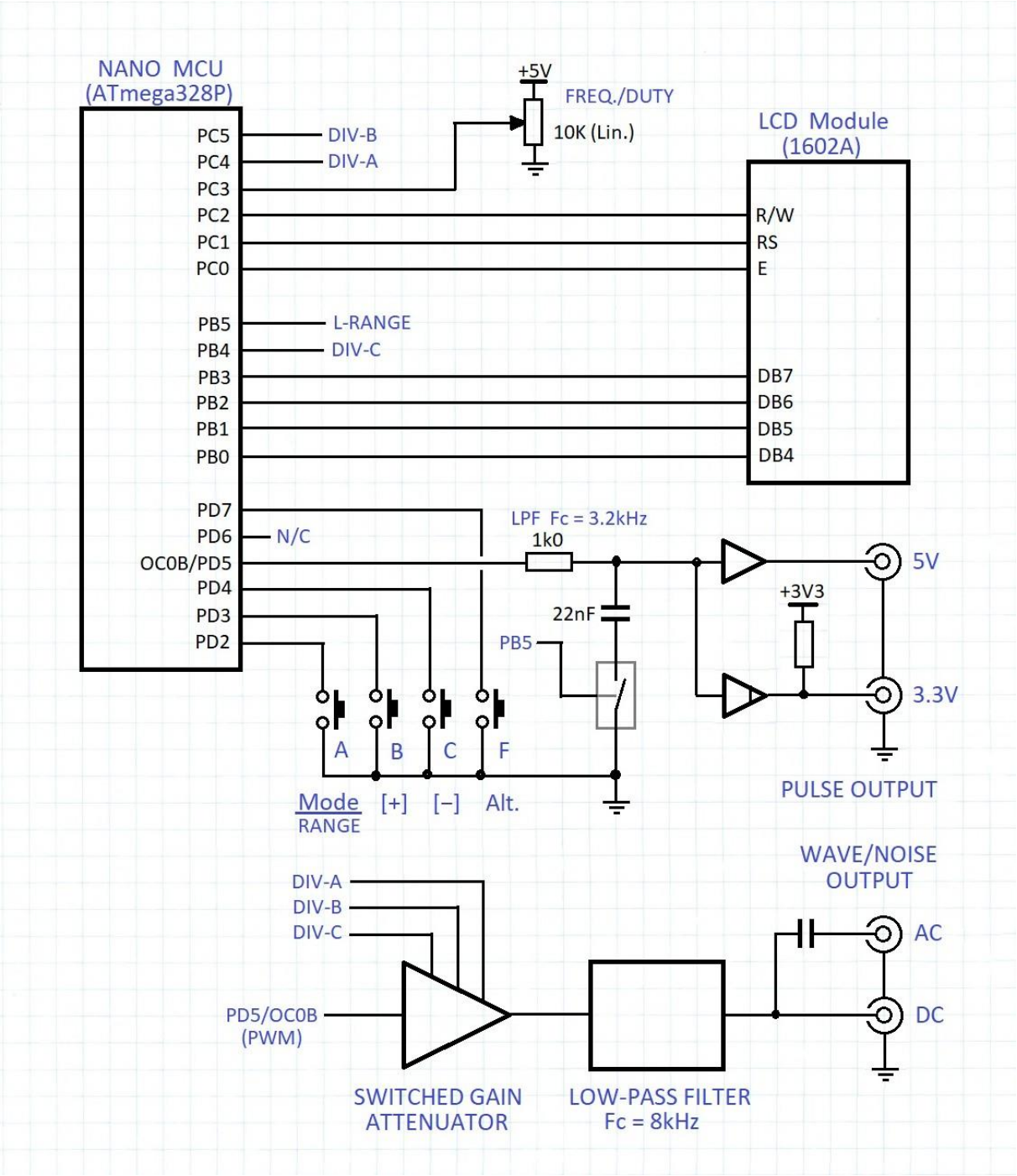
The project is based on the Arduino Nano v3 micro-controller development board plus a 2-line x 16-character LCD module (1602A) with LED back-light, four push-buttons and a potentiometer feeding into an ADC input. The pot controls either signal frequency or pulse duty, depending on the selected output mode. Output connectors are panel-mount RCA phono sockets.

The design makes use of an on-chip timer peripheral in the Nano AVR micro-controller (ATmega328P) to generate variable-duty pulse waveforms on an output pin.

In "WAVE" mode, a 32kHz PWM output is used to generate signals over the audio range (up to 8kHz) with a selection of wave-shapes, i.e. sine, triangle, square and sawtooth. These signals are produced by a "wave-table oscillator" algorithm in the firmware. The PWM functions as an 8-bit DAC (digital-to-analog converter) giving an amplitude resolution of 0.4% of full-scale.

The 32kHz PWM "carrier" frequency is removed from the audio output signal using a 3rd-order low-pass analog filter with a cut-off frequency about 8kHz and attenuation slope of −18dB per octave. The filter is built around a single op-amp (i.e. one half of a MCP602).

A full description of the workings of the wave-table oscillator code and PWM DAC can be found in the section "How it Works" further on.



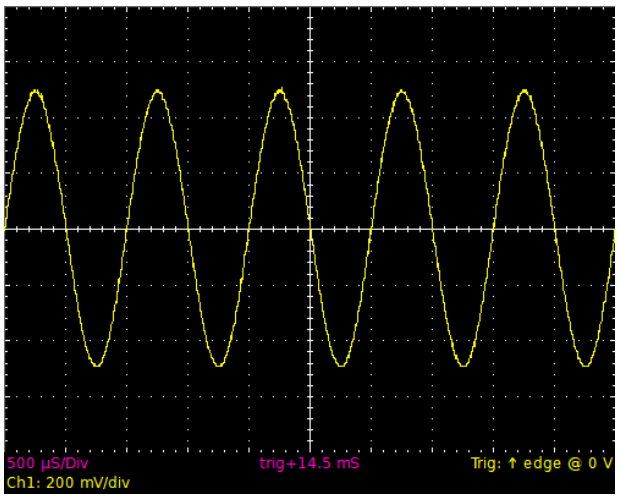
Simplified schematic diagram

The "WAVE mode" output level is set to one of six voltages using a switched-gain attenuator circuit interposed between the PWM output pin and the op-amp low-pass filter. The attenuator is implemented with 3 analog bilateral switches (74HC4066) controlled by 3 digital output pins on the micro-controller, plus an op-amp buffer (½ MCP602).

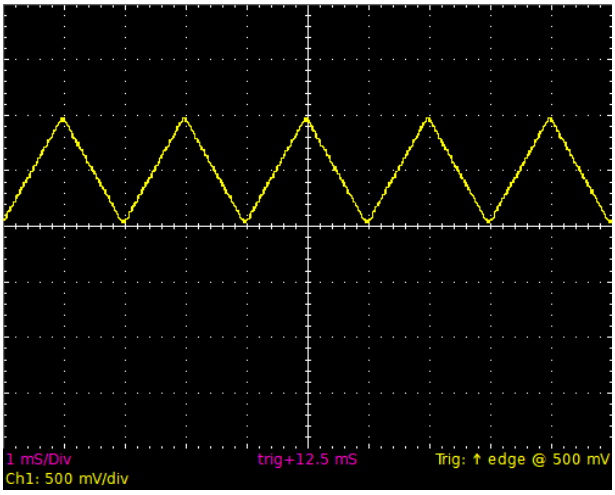
The "Low-frequency PULSE" mode (1Hz ~ 1kHz range) uses a similar method to the "WAVE" mode to generate a pulse waveform with variable duty. The PWM frequency is fixed at 32kHz. A simple first-order RC filter with a cut-off frequency at 3.2kHz removes most of the 32kHz PWM "carrier" to produce a pulse waveform. This is fed into a buffer (74HC125) which eliminates all residual 32kHz ripple, speeds up the pulse edges and improves the output drive capability.

In the “High-frequency PULSE” mode (1kHz ~ 4MHz range), the MCU timer module is configured to generate a variable frequency, variable duty pulse train directly on the timer output pin. The RC filter is disconnected by opening the bilateral switch (¼ 74HC4066). The output signal is buffered using a couple of gates wired in parallel (74HC125 quad tri-state buffer IC). This provides a “high current” drive capability (20mA sink or source) for the 5V pulse output.

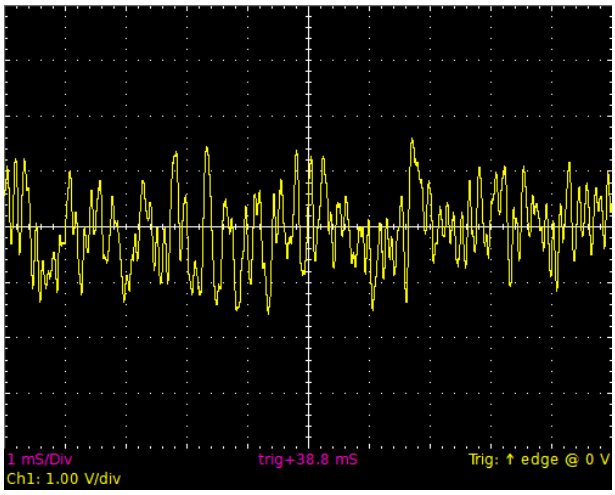
The other 2 gates in the buffer IC are wired with a 1k pull-up resistor to give a 3.3V output level. The pull-up idea is a compromise which results in slow rise-times, more apparent at the upper end of the frequency range. In hindsight, it might have been better to include a logic-level translator IC in the circuit. You can always add one if required.



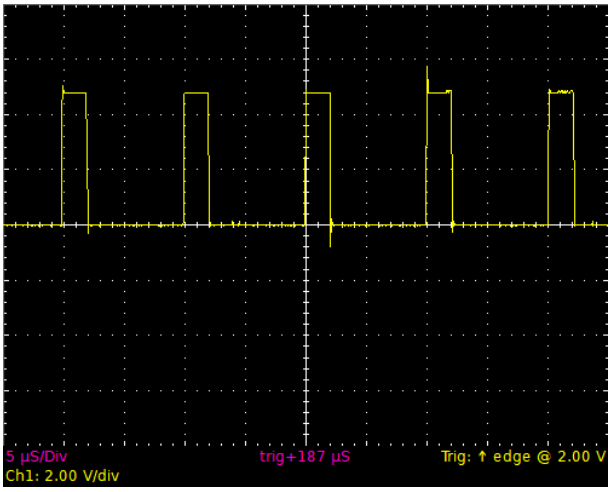
Sine-wave 1kHz, 1Vp-p AC out



Triangle wave 500Hz, 1Vp-p DC out

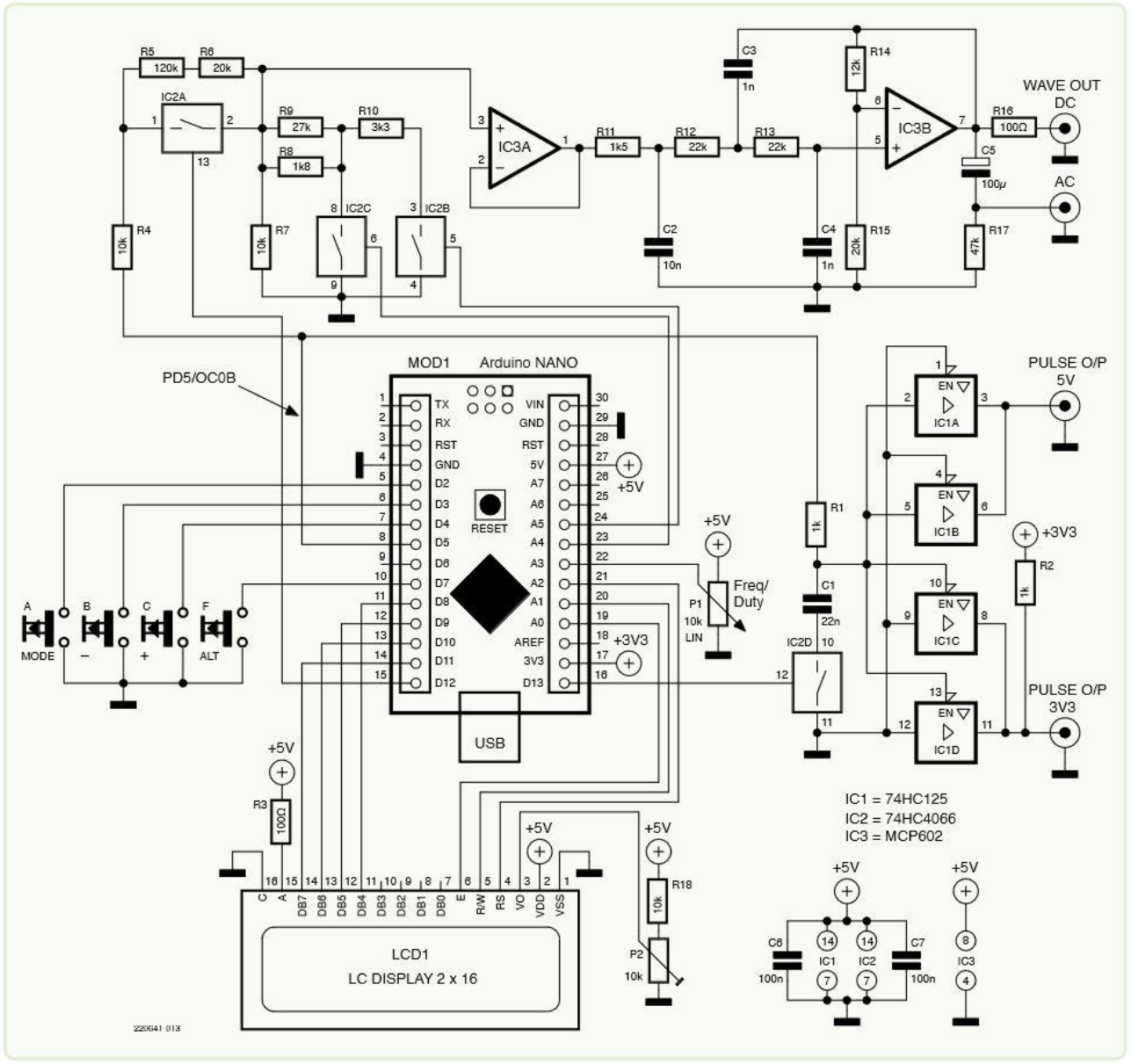


Noise, unfiltered, 4Vp-p AC out



Pulse mode, 100kHz, Tpw=2µs, 5V out

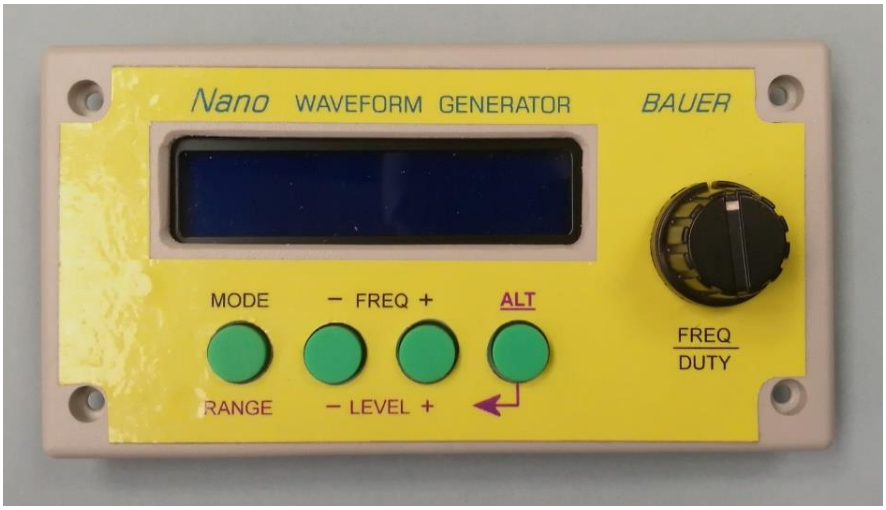
Examples of waveforms produced by the Nano WGM



Schematic diagram (courtesy of Elektor Magazine)

Control Panel Operation

The desired output waveform is selected by scrolling through the available options with the MODE button. Refer to the "Functional Specification" for implemented modes, frequency ranges and output level options.



In SINE-WAVE, TRIANGLE, SQUARE and SAWTOOTH modes, the FREQ/DUTY potentiometer may be used to adjust the output frequency (F_O) over the range 50Hz to 5kHz.

In NOISE mode, the pot is used to set the noise filter cut-off frequency (50Hz ~ 3.2kHz).

In SINE-WAVE, TRIANGLE, SQUARE, SAWTOOTH and PULSE modes, the FREQ+ and FREQ– buttons may be used to scroll up or down through a range of preset frequencies. This method of selecting the signal frequency gives a more accurate setting than the potentiometer. Also, the fixed frequency values were chosen so that the output signal will have no aliasing artifacts.

In PULSE mode, the pulse duty (*aka* "pulse width", T_{PW}) is adjusted using the potentiometer. The duty value is quantized to give a "rounded" accurate setting, i.e. it is not continuously variable. The resolution of duty value is dependent on the frequency setting. At high frequencies, the duty resolution is coarse; at lower frequencies it is finer. Pulse duty is displayed as a percentage of the period, also shown as pulse width (duration, T_{PW}) in appropriate units.

The "ALT" button when held pressed enables alternative functions for the other 3 buttons.

To select the frequency range, hold down the ALT button and press the RANGE button to toggle between High and Low ranges.

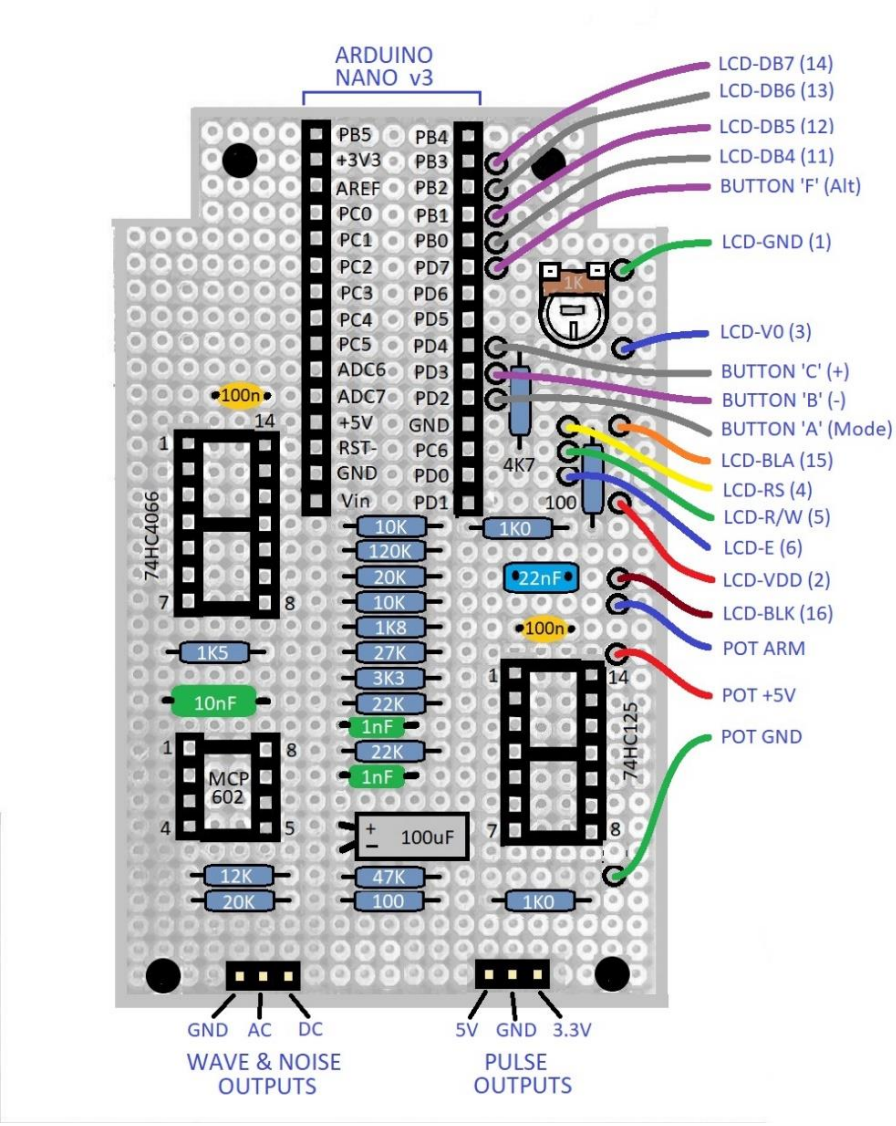
To adjust the output signal level in all modes except PULSE, hold down the ALT button and press the LEVEL+ or LEVEL– button to scroll through the available voltage settings.

Construction

The circuit is built on a piece of prototyping board of the sort that has a matrix of isolated pads on a 2.54mm (0.1 inch) grid. Ideally, use a double-sided plated-through board made of fibre-glass material. These boards are readily available in various sizes from online suppliers.

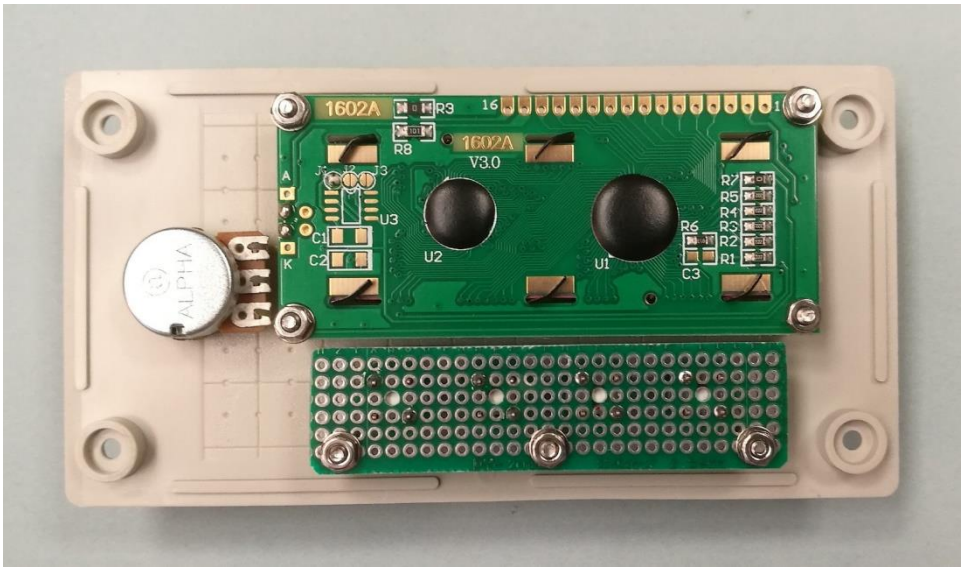
The board should be cut to measure 55 x 90 mm (2.2 x 3.5 inches, 21 x 34 pads). If your enclosure has posts in each corner for screws to fix the top cover, rebate two corners of the board so that the short side fits flush with one end of the box. The Nano USB socket is located on this side.

Drill four 3mm mounting holes in the corners. The board is fixed to the bottom of the enclosure on top of four 6mm spacers with 3mm machine screws and nuts. It's a good idea to glue the spacers inside the enclosure so that they stay put while screws are inserted.



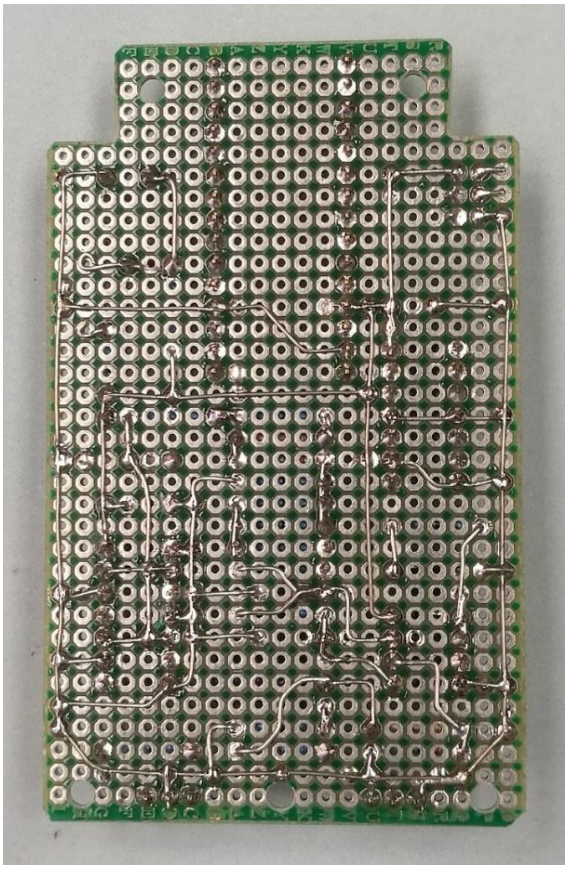
Mount the LCD module on the enclosure top cover using plastic spacers and machine screws. Print a photo of the top panel in actual size to use as a template for marking locations of the LCD window cutout and the hole for the pot spindle.

Four push-buttons are soldered onto another piece of proto board which is also mounted inside the top cover, below the LCD module. Before soldering the buttons on, drill 2mm holes in the board at the centre location of each button. The board itself can then be used as a template for marking the locations of the holes in the enclosure top cover.

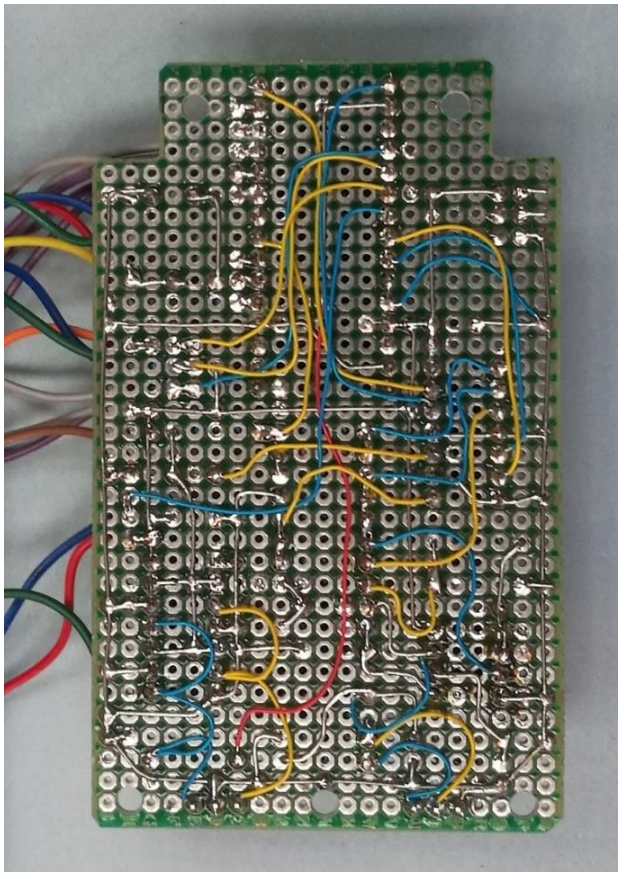


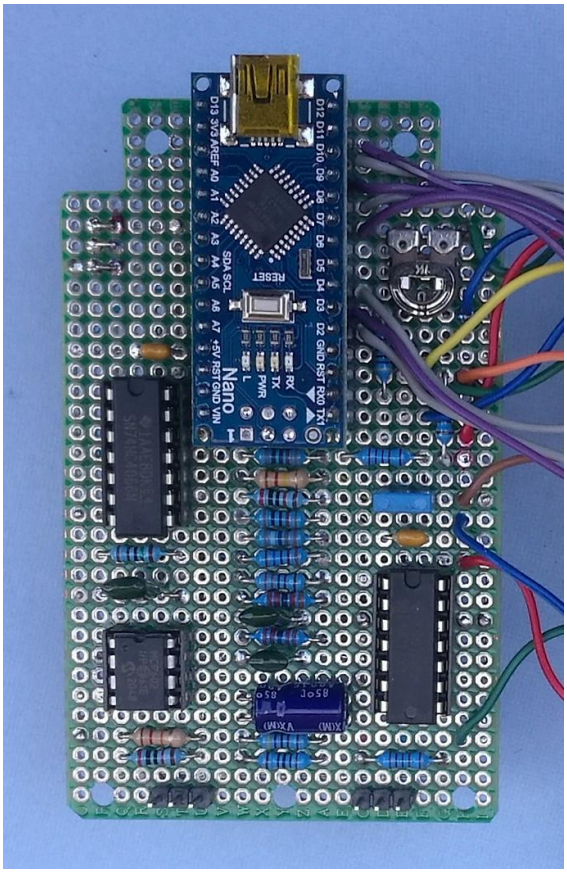
Begin by soldering two 15-way female strip headers (0.1-inch pitch) in place for the Nano micro-controller module. Solder IC sockets for the 3 IC's and two 3-way pin headers for connections to the output sockets. (Note: IC sockets are optional, but they make it easier to replace an IC if any become damaged.)

Proceed to solder remaining components in place, as shown in the board layout picture. Don't fit the Nano MCU module or IC's in their sockets until all wiring is completed and checked. Keep offcuts of wire trimmed from component leads. These come in handy for short wiring hops.



Using the photo of the underside wiring as a guide, run bare tinned copper wire around the perimeter of the board, following the outermost pads, steering clear of mounting holes. This is the GND rail. Complete all connections to GND with bare tinned copper wire, as shown in the diagram. A wire gauge similar to resistor leads (~24AWG, 0.5mm) is preferable.





Most of the connections to Vcc (+5V) can also be made using bare wire. Otherwise, use hook-up wire on the underside, or a wire link on the top side, in either case using insulated wire.

Some of the signal wiring may be done with bare wire, where wires do not cross. Remaining signal wiring is best done with 30AWG single-strand insulated wire-wrap wire. Kynar insulation is preferable, but it is expensive and becoming hard to find these days. PVC insulated wire is acceptable and much cheaper.

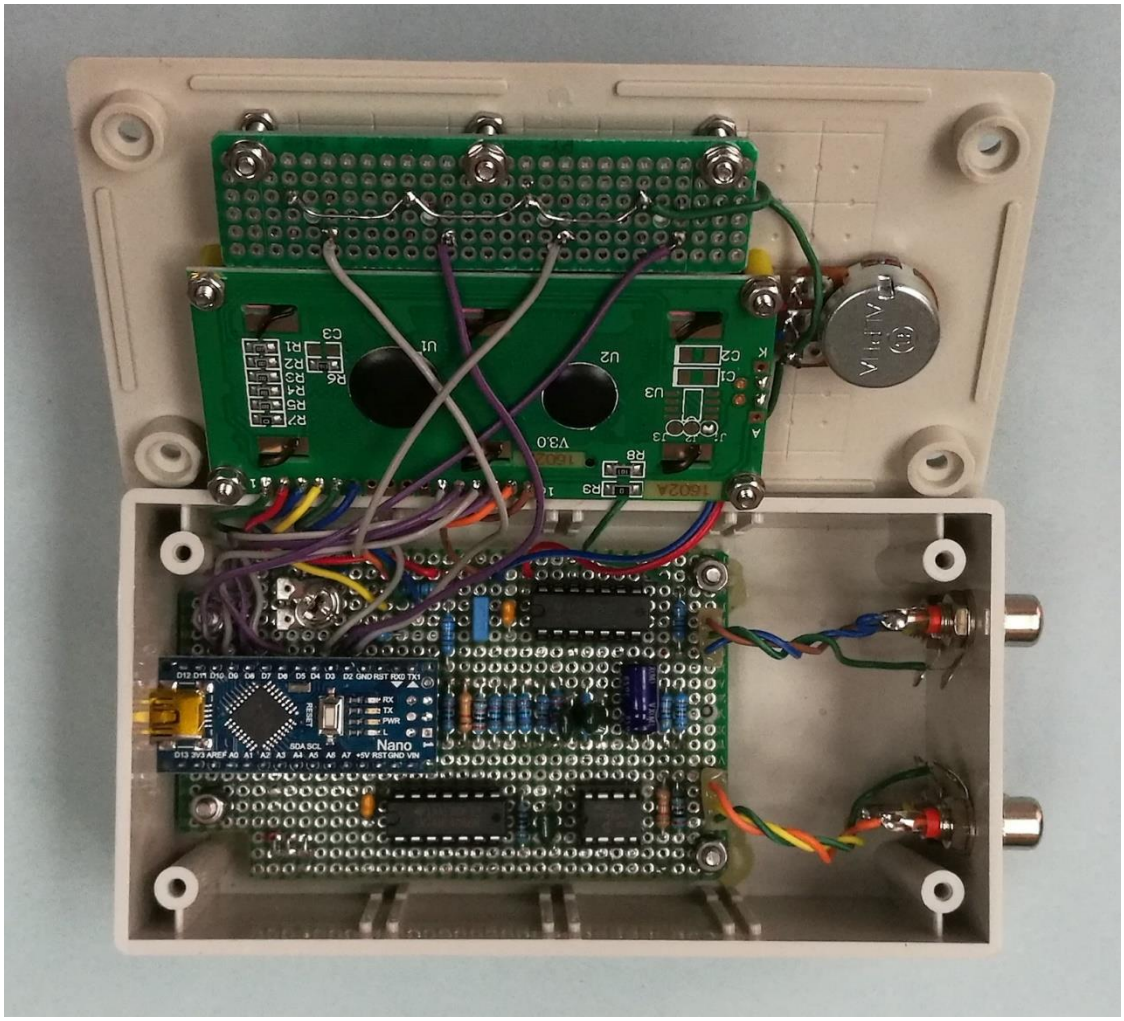
Carefully following the schematic diagram, complete the signal wiring. Cut hook-up wires a few mm longer than direct point-to-point distance, so they can be moved out of the way when adding further wires. Strip about 1.5 ~ 2mm of insulation from each end, taking care not to nick the wire as this will likely cause it to break off at the solder joint.

Where there is more than one wire to be soldered to the same pad, they need to be held in place and soldered together. This may be made easier by stripping off 4mm of insulation from each wire, then twisting the bare ends together and applying solder. Avoid wiggling soldered wire links more than necessary, because they break off easily.

When done wiring, it is highly recommended to check for point-to-point continuity and short-circuits using a multi-meter on the low Ohms range, or beeper. In particular, check that Vcc is not shorted to GND!

After the circuit board is well checked against the schematic diagram, connect the LCD module, push-buttons, and control pot to the board. The easiest and most reliable method is to solder hook-up wires directly to the board. Use a light-duty (13 x 0.12mm) multi-strand hook-up wire for board-to-board connections. Multi-coloured ribbon cable of the sort used for IDC crimp connectors makes good hook-up wire. It is easy to split into separate wires.

Then fit the Nano MCU module and IC's in their sockets. Set the LCD contrast adjust trim-pot to about half-way. Mount the circuit board in the box as shown in this photo…



Now it's time for the "smoke test". Connect your Nano to a 5V DC power source via the USB port. The LCD backlight should be on. If not, remove the power cable and trouble-shoot the problem.

Check with a DMM that +5V is present on all required Vcc pins, starting at the MCU module. If the DC voltages are correct and you don't see any smoke, your board should be ready for testing with firmware, as follows…

Firmware Installation

The firmware was developed using Microchip Studio for AVR and SAM Devices (formerly Atmel Studio). The primary reason for choosing this IDE (Integrated Development Environment) instead of Arduino is that the waveform generator hardware design is not compatible with available Arduino code libraries. In particular, the 1602A LCD interface scheme (MCU I/O pin assignment) does not appear to be supported by an Arduino library.

Programming the target device (ATmega328P MCU) can be achieved without Microchip Studio and without any hardware programming tool. The Nano module has an on-board USB-serial bridge device and a resident AVR bootloader. A Windows PC application called "**AVRdude**" communicates with the bootloader via USB to program firmware into the Nano's MCU flash memory. The same system is used by the Arduino IDE software.

Hence you need to download some files to run "avrdude" on Windows. The best place to download the files is GitHub, here: https://github.com/mariusgreuel/avrdude/releases. There should be 3 distribution files: "avrdude.exe", "avrdude.conf" and "avrdude.pdb".

Connect your Nano wave generator to a USB port on your PC.

Open Windows "Device Manager" utility and click on "Ports (COM & LPT)". You should see the Nano USB-serial device listed. Note the number of the associated "COM" port and use this number when editing the avrdude command line.

The programming operation is best performed using Windows "Command Prompt". A single command line does the trick. However, the required avrdude command line is quite long and it needs to be customised to suit your particular PC software setup and COM port allocation.

Copy and paste (or type) the command line below into a text editor such as Notepad. Be sure to delete any line feeds, so that the command is all on a single line. A file "**Program Nano.bat**" containing this command may be already available to download from wherever you found this article. Select "Word Wrap" from Notepad's "Format" menu to make it more readable.

```
avrdude.exe -C avrdude.conf" -p atmega328p -c arduino -P COM4 -b 115200
-U flash:w:nano-wave-gen-v1.5.hex:i
```

Some of the yellow highlighted fields will need to be edited to suit your PC setup and the firmware revision to be programmed.

Replace `COM4` with the actual COM port your Nano board is connected to, as you found using Windows Device Manager. Annoyingly, the allocated COM port can change when the USB cable is unplugged and reconnected. This is because a USB-serial connection is made via a "virtual serial port" – not a physical port. (Remember when PCs had one or more real serial ports using a 9-pin D-shell connector? And the COM port numbers stayed fixed!)

Next, download the latest version of the Nano Wave-Generator firmware from wherever you found this article. The file will have a name like "nano-wave-gen-v1.5" with extension "**.hex**". Edit the avrdude command line so that the hex file name (highlighted) is identical with the one downloaded. Save the edited command file as "**Program Nano.bat**".

If you need to make changes to the command file, right-click the file-name and select "Edit" from the drop-down menu, or open Notepad first, then open the file from within.

Create a new folder named “Nano Wave Gen” on your PC local drive (C:). Copy the 3 downloaded avrdude files and the firmware hex file “nano-wave-gen-v#.#.hex” and the command file “Program Nano.bat” into the folder.

Open Windows Command Prompt from the Start menu and proceed as follows...

Navigate to the folder into which you copied the hex file and the command file. For example, at the command prompt (shown here in purple), type the text shown in blue:

```
C:\Users\user> cd C:\Nano [Tab][Enter]
```

... where [Tab] means hit the Tab key. This invokes “auto-completion” of file and directory names, so you don’t need to type in the full name (Nano Wave Gen), assuming there is no ambiguity. Windows should respond with a new prompt…

```
C:\Nano Wave Gen> _
```

Execute the avrdude command... Enter “Program Nano” at the prompt…

```
C:\Nano Wave Gen> Program Nano [Enter]
```

Assuming all goes well, avrdude will output a message confirming that the programming operation was successful.

If you have trouble using avrdude, try searching online forums offering help.

Note: Some cheap Nano board clones use a non-standard Baud rate for the serial bootloader, typically 57600 baud. If avrdude outputs an error message, try using a different Baud rate. In the avrdude command file, replace “`-b 115200`” with “`-b 57600`”.

How the Waveform Generator Works

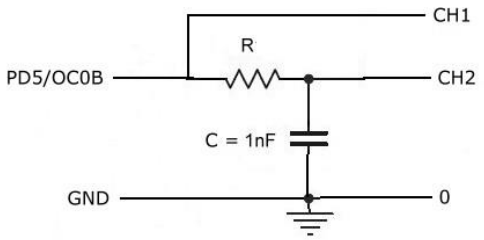
The technique employed is called "wave-table synthesis". A wave-table is an array of numbers representing one full cycle (1 period) of a waveform to be reproduced. The array elements are instantaneous signal amplitude values called "samples".

Samples are output at a fixed rate called the "sample rate", the frequency of which must be much higher than the maximum frequency component (“harmonic” or “partial”) in the output signal. A rough rule of thumb is 4 times higher. So, to generate audio signals up to 10 kHz, we would need a sample rate of 40 kHz. For convenience, due to the Nano MCU clock rate (16MHz), this application uses a lower sample rate, 32 kHz.

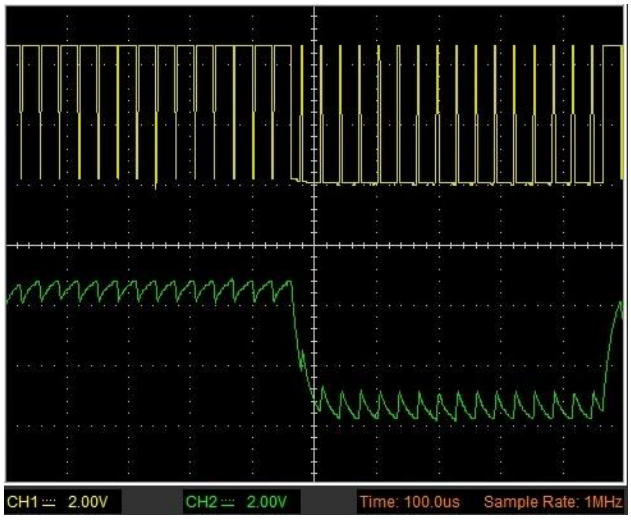
The firmware uses the ATmega328P on-chip Timer/Counter module TC0 in PWM mode to generate arbitrary waveforms in the audio and sub-audio frequency range. The duty of the PWM output signal is modulated by sample values taken from the wave table. The signal is passed through a low-pass filter to obtain a “short-term average” voltage that follows the shape of the waveform stored in the wave-table, albeit with a phase lag.

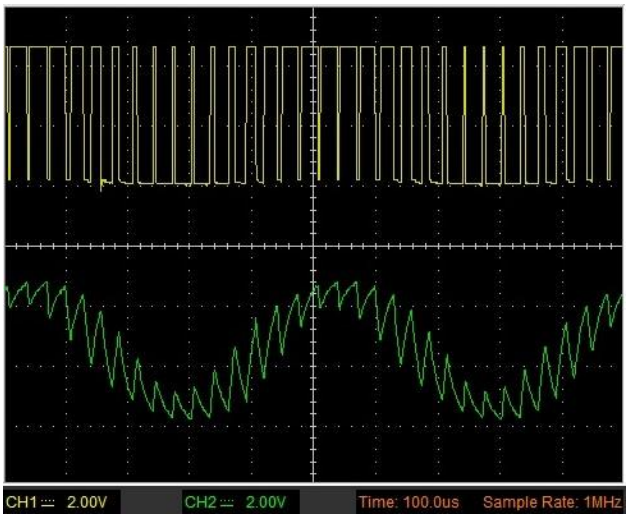
The filter cut-off frequency is chosen so that the 32kHz pulse frequency is removed as far as practical, leaving only the desired audio frequency components present in the output signal. The micro-controller PWM signal generator combined with the low-pass filter implements a digital-to-analog converter (DAC), thereby saving the cost of a DAC chip.

Operation of the “PWM DAC” is probably best explained using a simple first-order RC filter which only partially attenuates the 32kHz PWM carrier frequency.



In the oscilloscope screen-shots below, the top trace (CH1) is the raw PWM signal generated by the MCU (pin PD5/OC0B). The lower trace (CH2) is the filter output.





The left picture shows signals obtained with wave-table data for a square-wave. In this example, a full cycle consists of 32 samples at intervals of 31.25us, giving a period of 1ms, hence a frequency of 1000Hz for the output waveform.

Note that the PWM duty is near 100% during the first half cycle and near zero during the second half cycle. When the pulse duty is near 100%, the filter capacitor remains nearly fully charged (5V) and when the duty is near zero, the capacitor voltage drops to near zero. The bottom trace shows the capacitor charge and discharge curves. This manifests itself as a sawtooth-shaped ripple superimposed on the (almost) square-wave output signal.

The right-side 'scope shot shows signals obtained with wave-table data for a sine-wave. In this example, a full cycle consists of 16 samples x 31.25us, giving a period of 500us, hence a frequency of 2kHz for the output waveform. Note that the PWM duty is changing smoothly, following the sinusoidal curve. The sine-wave "zero crossings" occur when the PWM duty is 50%. The output wave peaks when the duty is near 100% (positive peak) and near zero ("negative" peak). Because the PWM signal is unipolar (0 to +5V) the output waveform is biased to 2.5V.

If we wish to attenuate the PWM "ripple" to an acceptably low amplitude, we need a better filter.

To keep the filter design as simple as possible, the cut-off frequency must be well below the PWM pulse frequency. A 3rd-order (3 pole) filter with cut-off at 8 kHz was chosen to obtain a good compromise between circuit complexity and performance. The filter attenuates the 32kHz PWM "ripple" to about 36 decibels below the audio signal peak amplitude. A 36dB reduction is about 2%, which is acceptable for this application.

See 'scope shots elsewhere in this article showing waveforms obtained with the 3rd-order filter.

Timer Functions in the Firmware

The function `TC0_setup()` initializes Timer-Counter TC0 in "dual slope" (*aka* "phase correct") PWM mode to generate a variable-duty pulse waveform on pin OC0B (= PD5).

Timer TC0 Output Compare register 'A' is used to generate a periodic interrupt at the sample rate, 32 kHz, so the interval between IRQ's will be 31.25 microseconds. The interrupt service routine (ISR) needs to fetch, process and output one sample every 31.25 microseconds. This is quite a challenge for a low-end 8-bit micro-controller, but with a clock frequency of 16 MHz, up to 16 AVR core instructions can be executed in every microsecond. Hence, up to 16 x 31.25 = 500 instructions can be executed in 31.25µs. Much can be done with 500 instructions without resorting to assembly language coding.

Of course, the CPU can't spend all its time in the ISR, otherwise nothing would get done in the main (background) loop, so we need to make sure that the ISR's maximum execution time is much less than the IRQ interval (31.25µs), preferably less than half of the IRQ interval.

The timer clock pre-scaler is disabled (N = 1) to get the highest clock rate. Assuming the CPU clock frequency is 16.0 MHz, register OCR0A is set to 249 for a timer period of 250 clocks (in each direction – up and down) so the PWM output frequency will be 32.0 kHz.

Output Compare B register (OCR0B) is used to generate a PWM output on pin PD5/OC0B. The PWM duty is varied in proportion to the sample amplitude, updated at the sample rate, i.e. every 31.25µs, by the timer Output Compare A interrupt service routine (ISR).

When the counter register (TMR0) reaches its "TOP count" (OCR0A = 250), the count direction is reversed and an IRQ flag (OCA) is raised. At the same time, the PWM output pin PD5/OC0B is set High (1).

When the count register (TMR0) matches the Output Compare B register value, the PWM output signal on pin PD5/OC0B is reset LOW (0) automatically. The PWM pulse duty is therefore proportional to the OCR0B register value. The duty obviously cannot exceed the period, so the maximum duty (100%) is obtained by writing 249 into register OCR0B.

For a more detailed explanation of PWM waveform generation using an AVR micro-controller, please refer to the ATmega328P datasheet, in the section on 8-bit Timer/Counter TC0.

Wave-table Oscillator Algorithm

The wave-table oscillator algorithm works by fetching samples from a wave-table at a fixed sample rate, 32 kHz. The frequency of the output signal is determined by the "distance" (phase angle) between points taken from the wave-table. The shorter the distance (angle), the lower the output frequency. When the table index of the next sample to be fetched goes beyond the end of the table, the index is adjusted so that it “wraps around” to a point within the table maintaining the correct sample "distance".

Let's call the distance between sample points the "Phase Angle Step". For most practical purposes, to obtain the required accuracy of signal frequency, the Phase Angle Step needs to be a real number, i.e. having both an integer and fractional part. Floating-point arithmetic would make the task easy, but this is too slow for the Nano AVR micro-controller without floating-point hardware in the CPU, so fixed-point arithmetic is used instead.

Fixed-point arithmetic uses fast integer arithmetic with integer words (or long words) composed of two “bit fields” representing the “integer” and “fractional” parts of a number.

A 16-bit fixed-point number could be composed of an 8-bit integer part and an 8-bit fractional part. For unsigned numbers, the range would be 0.0 ~ 255.99 (approx. decimal equivalent). For signed numbers, the range would be −127.99 ~ +127.99. The resolution (accuracy) is 1/256 = 0.004 (approx.) which may be adequate for some low-precision applications.

A 32-bit fixed-point number is often composed of a 16-bit integer part and a 16-bit fractional part. For signed numbers, the range is roughly −32,000 ~ +32,000 (decimal equivalent). The resolution (accuracy) in this case is roughly 1/65000 = 0.00001, which is more than adequate for our waveform generator. The accuracy of output signal frequency is limited only by the crystal on the MCU system clock.

In the Nano wave-table oscillator, the Phase Angle Step and Phase Angle (wave sample point) are represented by 32-bit fixed-point numbers. The integer part of the Phase Angle is used as an array index to fetch a sample point from the table.

The formula relating “Phase Angle Step” to Oscillator Frequency is:

PhaseStep = OscFreq x (TableSize / SampleRate)

where:

PhaseStep is the distance between sample points in the wave-table
OscFreq is the required output frequency (Hz)
TableSize is the total number of samples in the wave-table
SampleRate is the sampling rate (32,000Hz)

Although wave-tables may be any arbitrary size, within the constraints of program memory, the table size should be consistent with the output signal resolution. As a rough “rule of thumb”, the size should be around the same order as the maximum value of the samples in the wave-table. For example, if the output sample values are 8-bit unsigned numbers, a suitable wave-table size would be 2^8 = 256 samples. A larger table size gives the same benefit as interpolation between sample points, i.e. reduced waveform distortion.

Signal Aliasing

If you have studied Digital Signal Processing (DSP), you may be familiar with the term “aliasing” and its causes and effects. Aliasing is the distortion of a sampled waveform which results when a frequency component (i.e. “harmonic” or “partial”) in the analog input waveform is too high in

relation to the sampling rate. The distortion manifests itself as unwanted frequency components being introduced into the sampled signal.

For example, if an input signal contains a harmonic of frequency 16kHz and this signal is sampled at a rate of 20 kilo-samples per second (20kHz), there will be components in the (reconstructed) output signal at the sum and difference frequencies, i.e. 4kHz and 36kHz. The 4kHz component is well within audible range and would be an unwanted artefact.

Pure sine-waves at frequencies below half the sample rate will not suffer any aliasing effect. Waveforms rich in high-order harmonics, for example square, pulse and sawtooth waves are very susceptible to aliasing.

Likewise, aliasing occurs with wave-table oscillators when a waveform represented by sample points from a wave-table contains frequency components above half the sample rate, except where the ratio of output frequency to sampling frequency is a rational number.

Careful choice of wave-table size relative to the sample rate can eliminate aliasing at certain output frequencies. In the Nano waveform generator, the table size (640 samples) and selection of fixed output frequencies was chosen so that aliasing does not occur.

Additional Material and Resources

Essential

- Schematic Diagram (pdf)
- Front Panel Decal picture: “Nano Wave Gen panel decal (300dpi).jpg”
- Firmware (Hex object-code file): “nano-wave-gen-v#.#.hex”
- AVRdude command file: “Program Nano.bat”
- AVRdude software app for Windows (download from GitHub…)
  https://github.com/mariusgreuel/avrdude/releases

Optional

- Firmware Development Kit: “Nano Wave Generator FDK.zip”
- Microchip Studio (IDE) for AVR and SAM Devices
- Arduino Nano Reference Manual
- Atmel ATmega328P MCU datasheet (Microchip doc # 7810D-AVR-01/15)
- AVR “X-mini” Function Library (source-code): “lib_avrXmini.c”

Appendix A – Parts List

Description	Manuf. Part #	Qty
Prototyping board, 90 x 120 mm, pads on 0.1-inch grid		1
MCU module, Arduino Nano v3 (or clone) including 2 x 15-way pin headers		1
LCD module, 2-line x 16 characters, LED backlight, HD44780 (compatible) controller	1602A	1
Potentiometer, B-10K (linear), 16mm, panel mount		1
Knob for pot, 20mm diam. (approx.)		1
Enclosure, plastic, 130 x 70 x 40 mm (approx.)		1
PCB strip header, 40-way SIL 0.1”, female ccts		1
PCB pin header, 3-way SIL, 0.1” pitch		2
IC socket, 14-way		2
IC socket, 8-way		1
Trim-pot, horiz. mount, 10k linear		1
Push-buttons, PCB mount, snap action, SPST		4
Socket, RCA phono, panel mount		4
IC, quad analog switch, 74HC4066, 14-pin DIP		1
IC, quad tri-state buffer/gate, 74HC125, 14-pin DIP		1
IC, dual CMOS op-amp, MCP602, 8-pin DIP		1
Capacitor, Electrolytic, 100µF 16V		1
Capacitor, Polyester, 22nF 50~100V (pref. MKT)		1
Capacitor, Polyester, 10nF 50~100V		1
Capacitor, Polyester, 1nF (.001 µF) 50~100V		2
Capacitor, Monolithic ceramic, 100nF (.1 µF) 50V		2
Resistor, 100Ω, ½W 1% metal film		2
Resistor, 1k5, ½W 1% metal film		1
Resistor, 1k8, ½W 1% metal film		1
Resistor, 3k3, ½W 1% metal film		1
Resistor, 10k, ½W 1% metal film		3
Resistor, 12k, ½W 1% metal film		1
Resistor, 20k, ½W 1% metal film		2
Resistor, 22k, ½W 1% metal film		3
Resistor, 27k, ½W 1% metal film		1
Resistor, 120k, ½W 1% metal film		1
Hookup wire, stranded 13 x 0.12mm, ass’d colours (or multi-colour ribbon cable, 0.05” pitch, split)		2m
Wire-wrap wire, 30AWG, Kynar (or PVC) insul’n		2m
Tinned copper wire, ~24AWG (0.5mm)		1m
Cable, USB Type-A (host) to ‘MINI-B’ plug		1m
Hardware – screws, nuts, washers, standoffs, etc		

Appendix B – Programming the Nano with Microchip Studio

To customise the waveform generator firmware to suit your own requirements, first download and install Microchip Studio for AVR and SAM Devices (IDE) on your Windows PC or Mac. This is a free software application with superior features, but no more difficult to use than Arduino.

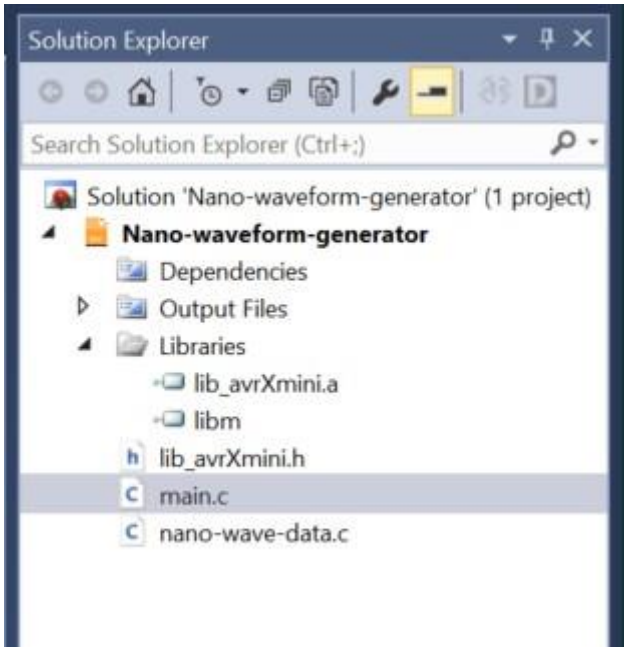
If you are not already familiar with Microchip Studio, it would be wise to work through an introductory tutorial aimed at developing applications for 8-bit AVR micro-controllers. The tutorial will probably assume a "target platform" using a Microchip or Atmel development board with a hardware programming tool, e.g. Atmel "AVRISP" mk2 or similar device. **You don't need a hardware programming tool** to load firmware into the Nano board. You can create a "software programming tool" in Microchip Studio. (See "Option 2" further down.)

Meanwhile, download the waveform generator firmware development kit (Zip folder) and copy all the files to a new folder named "**Nano-waveform-generator**" on your PC local drive, preferably in the projects folder created by Microchip Studio in your "Documents" library.

Connect your Nano wave generator to a USB port on your PC.

Next, open Microchip Studio and choose "Open Project…" from the Start Page. Navigate to the project folder you created and click the file-name "Nano-waveform-generator.**atsln**".

On the right side of the IDE window, find and click the tab labelled "Solution Explorer". The panel should look like the screen-shot below. Expand the Libraries list and ensure that the library "lib_avrXmini.a" is present. The library file contains "pre-built" functions to support various peripherals commonly used in ATmega328P MCU applications.



To edit a source file, e.g. "main.c" or "nano-wave-data.c", if the file is not already open in the editor window, click the file-name in the Solution Explorer panel. To keep the file open in the editor, click the "pin" icon in the file tab at the top RHS of the editor window. The file tab will move to the LHS of the window and remain open.

Feel free to peruse the library header file "lib_avrXmini.h", but **do not modify this file** unless you intend to make changes to the library. The C source code is freely available in case you want to modify any function(s) in the library or add more functions.

All going well, you are now ready to modify and extend the source code to implement your desired features. When done, build your "solution", fix any compilation errors and build again, then follow the instructions below to program the firmware code into your Nano board.

There are two options. In both cases, it is assumed that the avrdude distribution files can be found in a folder named "Nano Wave Gen" on your PC local drive.

Option 1: Use Windows Command Prompt as before

You can use the same method as described further back in this article under the heading "Firmware Installation". All you need to do is copy the Hex file generated by Microchip Studio into the folder "Nano Wave Gen". Every time you hit "Build Solution", Microchip Studio will replace the Hex file in the project folder, in a sub-folder named "Debug".

For example, if your project folder is named "Nano-waveform-generator", the Hex file will be written into the sub-folder "Nano-waveform-generator\Debug".

Before programming, rename the generated Hex file so that it is identical with the file-name contained in the Windows command file "Program Nano.bat".

Option 2: Create a "Programming Tool" in Microchip Studio

Click in the menu "**Tools/External tools**".

You should see a dialog box asking for some parameters, as follows...

In **Title**, write: `Program Nano` or any other name you prefer.

In **Command**, write: "`C:\Nano Wave Gen\avrdude.exe`"

In **Arguments**, write (all on one line):

```
-C "C:\Nano Wave Gen\avrdude.conf" -p atmega328p -c arduino -P COM4 -b 115200
-U flash:w:"$(ProjectDir)Debug\$(TargetName).hex":i
```

Replace COM4 (in the Arguments field) with the actual COM port your Nano board is connected to, as can be found in Windows Device Manager.

Tick the box: "Use output window". Click OK.

Done... You should see a new option "**Program Nano**" in the **Tools** menu.

After the code is built, it can be programmed into the Nano board simply by clicking on the item "Program Nano" in the Tools menu.

Note: Some cheap Chinese Nano board clones use a non-standard Baud rate for the serial bootloader, typically 57600 baud. If Microchip Studio outputs an error message when running the programming tool, try using a different Baud rate. Replace "`115200`" with "`57600`" in the Arguments field.