Goom

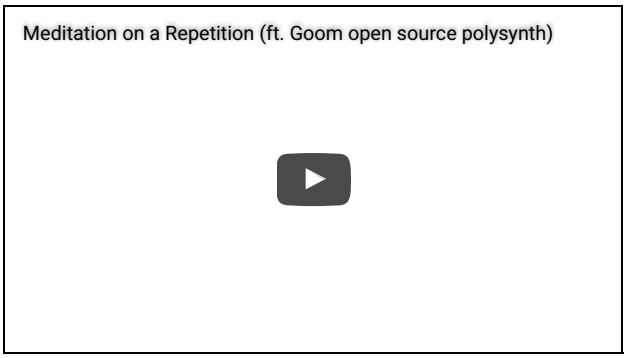
Quinapalus Home :: Things Technical :: Goom

Goom is a digital music synthesiser based on an NXP LPC1343 ultra-low-cost microcontroller. It broadly emulates the architecture of traditional analogue synthesisers, offers 16-voice polyphony and is fully multitimbral, and is controlled over a MIDI interface. The total cost of the basic components to make a fully-working synthesiser is just a couple of pounds; the (optional) 'knobs and switches' analogue front panel interface increases the total component cost by an order of magnitude or so, mostly accounted for by the potentiometers themselves.

Here's the demo track, recorded in a single take from Goom's analogue output without effects processing, dynamic range compression or other treatment. Sequencing was done using Rosegarden.

Audio not playing? Download it.

Thorsten Klose has ported Goom to run on the MIDIBox hardware platform using a rather faster STM32-series microcontroller. Justin L S Evans has made the following demonstration video, using a tablet running TB MIDI Stuff (*sic*) as a real-time controller and a template he designed himself.



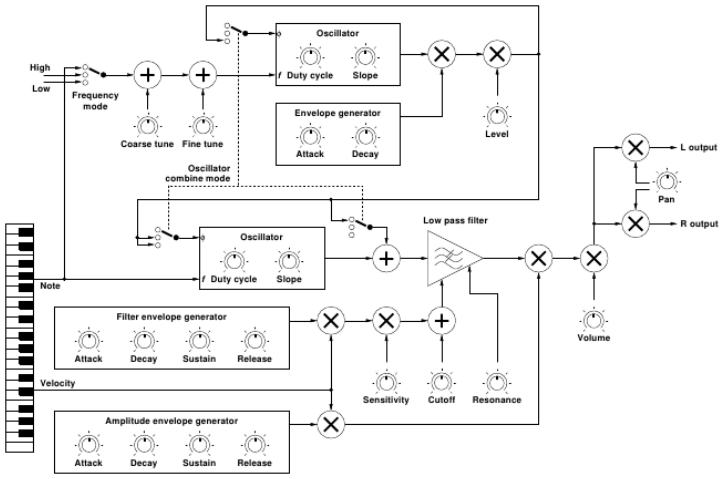
There is another demo of his here.

Features

- Sixteen-voice polyphonic
- Fully multitimbral (different patch on each MIDI channel)
- Analogue front panel patch set-up for MIDI channel 1
- Patch set-up using MIDI control change messages for channels 2 to 16
- Two oscillators per voice: sine, sawtooth, square, pulse and intermediate waveforms
- Oscillators can be mixed or combined using frequency modulation or frequency modulation plus feedback
- Three envelope generators per voice (one ADSR and one AD for amplitude, one ADSR for filter)
- Low-pass filter for each voice with resonance control
- Velocity scaling on amplitude and filter cutoff
- Stereo output with pan and volume control for each patch
- 24-bit digital-to-analogue converter

Voice architecture

The tone generation structure for each voice is shown in the diagram below. It broadly follows the conventional layout of an analogue synthesiser, but adds frequency modulation modes to increase the range of tone colours available.



One point of interest is that each oscillator waveform is controlled by a pair of continuous parameters rather than, for example, a multi-position switch. The waveform is divided into four parts: a rising slope, a flat period, a falling slope, and a final flat period. Each slope takes the shape of half a cosine wave: the first

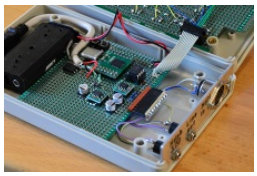
from cos –π to cos 0 and the second from cos 0 to cos π.

The first control determines the 'duty cycle', the ratio between the time taken for the first slope plus the first flat period to that taken for the second slope plus second flat period. The second control determines the proportion of the total cycle occupied by the flat periods.

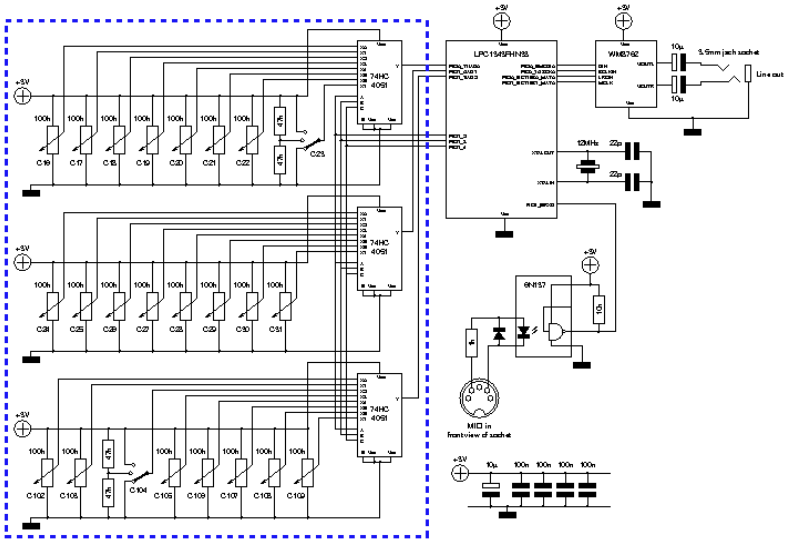
Together these two controls allow the generation of sine, square and pulse waveforms, and an approximation to a sawtooth waveform. Furthermore, a wide range of intermediate waveforms is also available. Very roughly speaking, the first control determines the presence of even harmonics (varying on a line from string to flute, if you will), while the second control determines the overall harmonic richness.

An upper limit is enforced on the slope of the waveform such that as the frequency increases the waveform approaches a sine wave. Since the waveform and its first derivative are continuous the result is in a worthwhile reduction in aliasing without having to resort to more computationally intensive techniques such as those that involve summing individual band-limited waveform fragments. A particular advantage is that much precomputation can be done at the control update rate to reduce the work that needs to be done at the output sample rate.

Hardware

The circuit (PDF here) is based around an NXP LPC1343 microcontroller, which includes an ARM Cortex-M3 core running at 72 MHz. The design also fits in the pin-compatible LPC1311, which has 8 kbytes of flash memory and 4 kbytes of RAM, but it has not been tested with this device. At the time of writing the LPC1311 is available for just over one pound in quantity one.



The only other components of any significance are the optocoupler required to isolate the MIDI input, and the audio digital-to-analogue converter (DAC). Remarkably high-resolution DACs are available remarkably cheaply: the prototype uses the Wolfson WM8762, a 24-bit DAC available for under one pound in quantity one. The prototype also included a headphone amplifier circuit based on the LM4881; alternatively compatible DACs with built-in headphone amplifiers are available. MIDI out can be implemented if desired by adding the standard interface circuit to the TXD pin of the microcontroller, and of course it is also easy to add MIDI thru.

The analogue front panel interface uses three 74HC4051 analogue multiplexers to simplify wiring and reduce the number of pins required on the microcontroller. The 'Cxxx' labels in the circuit diagram show the MIDI controller number to which each potentiometer or switch corresponds: see also the table below. This part of the circuit (marked by the dotted box in the circuit diagram) can be omitted if desired: if so, it is a good idea to ground the three analogue inputs it uses on the microcontroller.

Note that the circuit diagram does not show in-system programming circuitry, which you will almost certainly need to add. The LPC1343 can be programmed conveniently over its USB slave interface, or the serial port can be used: see the NXP documentation. The optocoupler connected to the RXD pin on the microcontroller has an open-collector output and so a serial programmer can be connected (temporarily) in parallel with it as long as no MIDI messages arrive during programming.

The LPC1311 does not have a USB interface.

The circuit runs on 3 V and can be powered from two AA cells. Current draw is around 50mA when not driving headphones.

If the circuit is run from a single supply then care must be taken with decoupling, especially around the optocoupler, to minimise the amount of digital noise that appears on the analogue outputs. A superior (but more expensive) solution is to use separately regulated supplies for the analogue and digital parts of the circuit, connected at a single ground point.

Software

The processor spends about 95% of its time in an interrupt routine computing output samples. Despite what many say, assembler code is considerably more efficient than compiled C, largely because of the possibility of making better use of registers and of conditional instructions; for this reason, the voice generation code is written entirely in assembler. An early prototype written in C was approximately a factor of two slower. Looked at another way, the C implementation would have allowed only eight simultaneous voices rather than sixteen.

Further optimisations include arranging data structures so that byte values appear earlier than half-words, and half-words appear earlier than words. This is because the maximum offset for byte load and store instructions that fit in sixteen bits is only 31 for bytes, but is 62 for half-words and 124 for words, and sixteen-bit instructions put less pressure on program memory access than 32-bit instructions. For the same reason, instructions using the low registers (R0 to R7) in general execute faster than instructions that reference R8 to R15.

A further interrupt routine serves the SSP module, which sends data to the DAC via its BCKIN and DIN pins.

MIDI messages received over the serial port are processed in the foreground code, with the update of control parameters postponed until there are no outstanding messages in order to ensure that no messages are lost.

System timing

The processor runs at 72 MHz. This clock is divided by eight to generate the master clock for the DAC (MCLK); by sixteen to produce the clock for the microcontroller's SSP module; and by 2048 to generate the word clock for the DAC (LRCIN). The DAC is therefore run in $256f_s$ oversampling mode and the audio sample rate is 72 MHz/2048≅35.2 kHz. A timer generates a further interrupt every four samples, i.e. approximately every 114 µs. The service routine for this interrupt calculates the next four output samples on each stereo channel for each of the sixteen voices and updates one envelope generator, and completes in approximately 108 µs.

The timer interrupts are carefully synchronised so that the control signals for the DAC have the correct phase relationship to one another and to the output of the SSP module.

The code occupies about 5.5 kbyte of program memory and uses just over 2.5 kbyte of RAM. The source code and hex file are available here. The code is open source, licensed under version 2 of the GNU GPL.

User Synthy at ctrlr.org has created a Panel that allows Goom's parameters to be edited, saved and reloaded. See here for more details.

MIDI implementation

The following MIDI messages are recognised. There is a one-to-one correspondence between MIDI channels and patches; the sixteen available voices are dynamically reassigned to channels as required (i.e., 'Omni off, Poly' mode).

Function	Recognised
Note number	0 to 127
Velocity: note on	1 to 127
Velocity: note off	No
After touch	No
Pitch bend	Yes, ±2 semitones
Control change	Yes: see below
Program change	No
System exclusive	0xf0 0x7d 0x01 triggers dump of knob settings on MIDI out as follows: 0xf0 0x7d 0x01, 24 bytes of settings, 0xf7
System common	No
System real time	No
Aux messages	All notes off only

Control change messages

Controller number	Function
16	Oscillator 1 waveform duty cycle
17	Oscillator 1 waveform slope
18	Oscillator 1 coarse detune
19	Oscillator 1 fine detune
20	Oscillator 1 attack time
21	Oscillator 1 decay time
22	Oscillator 1 output level
23	Oscillator 1 frequency mode: 0=key, 64=high, 127=low
24	Filter EG attack time
25	Filter EG decay time
26	Filter EG sustain level
27	Filter EG release time

28	Amplitude EG attack time
29	Amplitude EG decay time
30	Amplitude EG sustain level
31	Amplitude EG release time
64	Sustain pedal
102	Oscillator 0 waveform duty cycle
103	Oscillator 0 waveform slope
104	Oscillator combine mode: 0=mix, 64=FM, 127=FM plus feedback
105	Filter EG sensitivity: 64=none
106	Filter cutoff
107	Filter resonance
108	Volume
109	Pan

Similar projects

This project runs on a slightly lower-specification microcontroller. It offers four oscillators but is only monophonic, mostly because of the high sample rate used to try to reduce aliasing.

This project also runs on a slightly lower-specification microcontroller. It has three oscillators and is five-note polyphonic, but only has one filter shared among all the voices. This page describes some variations on the design, but it is not clear that any of them in fact exist.

There are several designs for monophonic synthesisers based on PIC and AVR microcontrollers, but most are rather limited in terms of waveform generation and filtering.

This page most recently updated Fri 8 Apr 16:12:19 BST 2016