PLUGGING VST AND DSSI SYNTHESIZERS INTO *ROSEGARDEN*

Audio and music production

PART 3 After building a simple synthesizer last month, Graham Morrison investigates other possibilities for sound generation.

Towards the end of last month's tutorial we got to the point where we could start to construct a project using the audio sequencer *Rosegarden*. While most of the article was concerned with building a basic synthesizer, the final stages involved recording the audio output from *AMS* into *Rosegarden*. This is a typical way of working with synthesizers and other audio applications, though with escalating processor speeds it's possible to run more and more virtual instruments without ever having to render the output to an audio file.

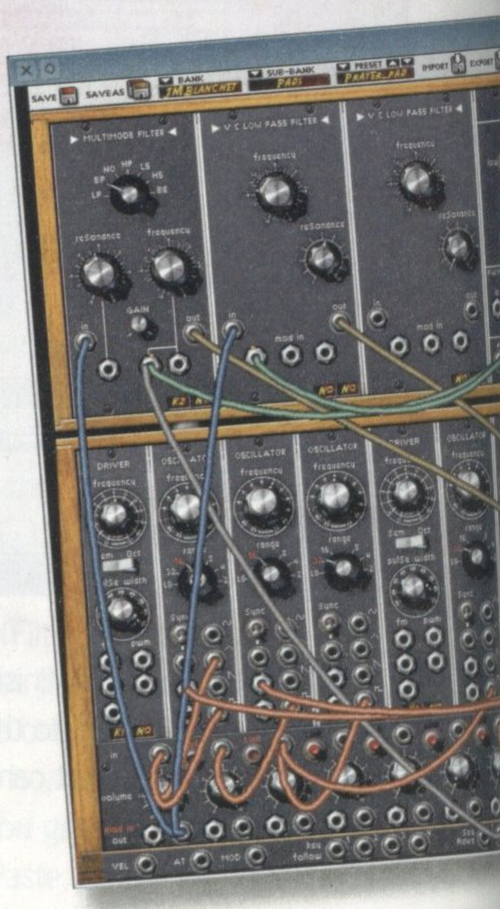
Most projects start with a glut of various MIDI sequences. Whether they're captured from an external controller, programmed manually or automatically generated by external MIDI processors (and it's usually a mixture of all three), creativity usually starts with these building blocks. Audio is also going under the knife, often being used as an elastic loop that can be wrapped and warped using software without going anywhere near either MIDI or a synthesizer. Despite this, there remains the need to generate the original source material.

Rosegarden originated as a MIDI sequencer, and it's on this foundation that the rest of its audio handling functionality has been built. The difference now is that you don't need external equipment to be able to generate any sound. As we've seen in the previous tutorials, software synthesis is becoming a viable alternative to owning the hardware, with the added advantage of being self-contained.

Arpeggiation

One of the best ways to generate MIDI data is by using an arpeggiator. Back in the dark ages of modular synthesis, an arpeggiator became the performance version of a step-sequencer. Where a sequencer typically featured 8 or 16 pre-defined notes played in order, an arpeggiator generated notes depending on the input, with various user configurable rules defining the relationship between the incoming notes and those produced by the arpeggiator. Modern varieties work on the same principle; and at its simplest, an arpeggiator mirrors the input notes to the output in either ascending or descending order, with a predefined value for delay and duration.

One of the most comprehensive arpeggiators available for Linux is by the prodigious Matthias Nagorni of *Alsa Modular Synthesizer* fame. Succinctly called *QMidiArp*, this application epitomises the less-is-more approach to user-interface design, effectively concealing its capabilities in a way that might deter the average user. A typical arpeggiator would generally provide plenty of hands-on control, featuring switches and sliders with immediate access to the most important parameters such as octaves, range and direction. This isn't the case with *QMidiArp*, which takes a typically Linux-like approach by using a small programming language to describe the input–output transformation. If you want to actually hear anything, it also needs to be MIDI-wired between an input from a MIDI device (such as an external MIDI keyboard or vkeybd) and something like the MIDI input of a synthesizer (routed to the audio outputs). An arpeggiator is usually a single instance, but with *QMidiArp*, it's possible to generate as many as you need all running concurrently.



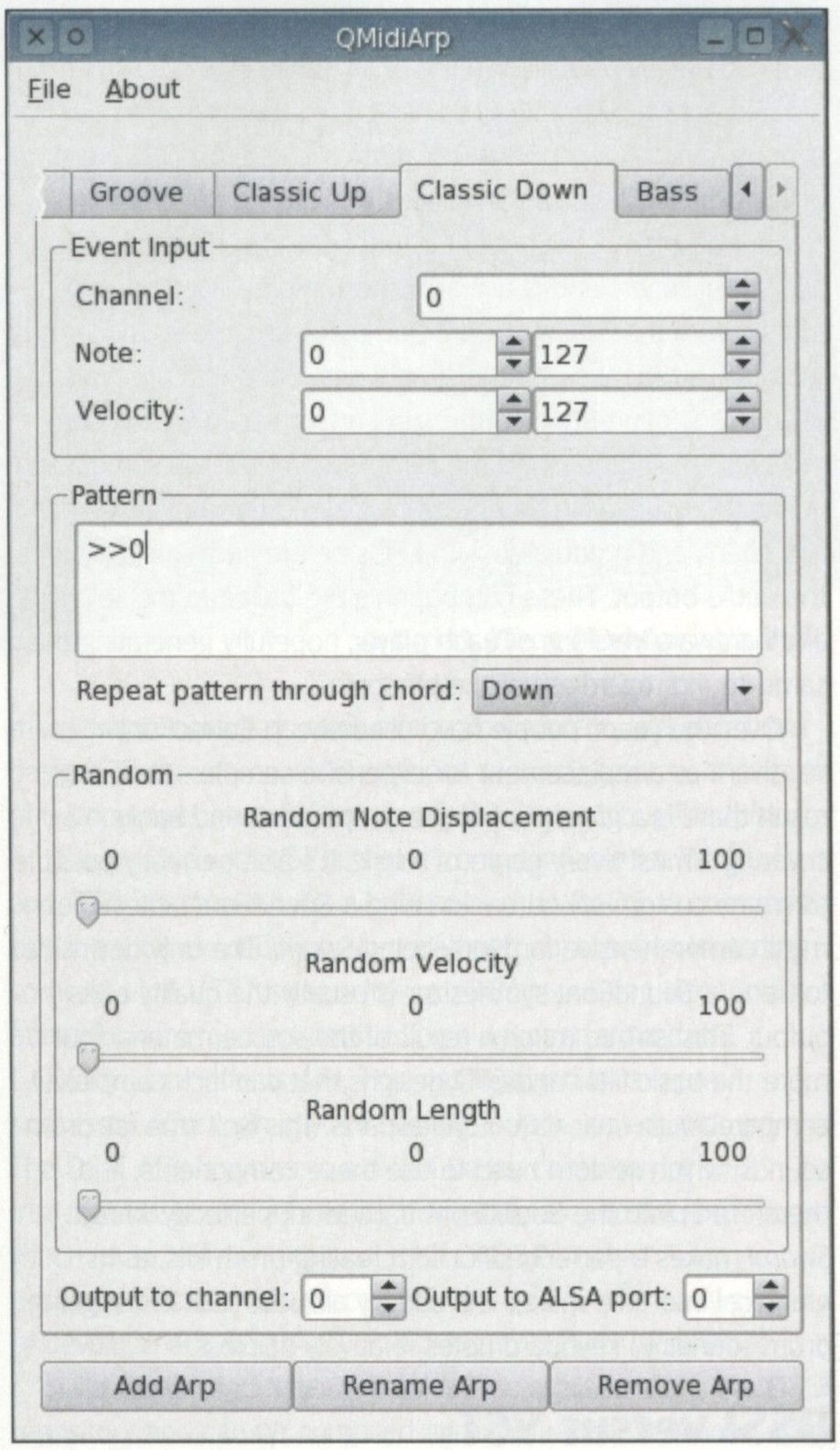
To generate a classic up–down arpeggiation in *QMidiArp*, press the Add Arp button and then enter 0 into the Pattern box. This is the hardest part to understand: the 0 tells the software to output the note at the first position on the input buffer (all incoming notes are added sequentially). The first note is removed from the buffer as it is played, moving the previously second note to the first position (First In First Out, or FIFO). For example, if the pattern 01 is entered the second note of the sequence would be repeated. This is because it's firstly played as the second in the buffer, and once more when it becomes the first. To make this clearer, with an input sequence of CDEF, with 0 for the pattern, the output would be CDEFCDEFCDEF. With a pattern of 01, the output would be CDDEEFCDDEEF. The final note isn't repeated because by the time it gets played there aren't any other notes in the buffer.

QMidiArp also has direction selection, which by default is set to Up – it works through the note buffer from the lowest note to the highest. For more complex arrangements, using more than one arpeggiator (especially with a combination of upward and downward sequences) can almost create complete tracks (think 80s electro-poppers Erasure or the *Streethawk* theme). For a good example of this, create three arpeggiators, with the first containing the simple pattern >>0 going up (>> quadruples the tempo). The second one's pattern should be up an octave (>>+0) but otherwise the same; and to generate a bass line, the third could be something like --0012 – this drops the sound down a couple of octaves and changes a three-note chord into an arbitrary four beats to the bar arpeggiation.

QMIDIARP COMMANDS

All you need to know to work this arpeggiator

0–9	Note indices in buffer
+	Octave up
-	Octave down
=	Reset to input octave
>	Double tempo
<	Half tempo
.	Reset to standard tempo
( )	Enclosed chord
/	Volume up by 20%
\	Volume down by 20%
d	Double length
h	Half length
p	Pause



Utilitarian, but *QMidiArp* sounds great.

While it's great to be able to play all this auto-generation live, the output becomes a lot more useful when recorded into a sequencer such as *Rosegarden*.

MIDI recording

Firstly, it's important to make sure that both the sequencer and the arpeggiator are set to the same tempo. This ensures that the arpeggiator notes can be made to synchronise with an entire *Rosegarden* project. In *QMidiArp* this is configured on the Settings page, in *Rosegarden* it's a little more complicated. It is possible to define changes in tempo at any point within a project, but changing a whole project's tempo is only possible at the beginning of the arrangement's timeline. To ensure that the current position is set to the start, press the Rewind To Beginning button then set the tempo either by double-clicking on Tempo in the Transport window (by default it's 120 beats per minute) or via the menu (Composition > Tempo And Time Signature > Add Tempo Change). For a moderately slow tempo, try something like 95.

To record *QMidiArp*'s MIDI output into *Rosegarden*, it firstly needs to be routed to *Rosegarden*'s MIDI input from *qjackctl*. To makes things clearer in *Rosegarden*, it's wise to rename the inputs and outputs from the Manage MIDI Devices dialog to reflect the *QMidiArp* connections and that of any other synthesizers. It's also essential to make sure that the Record Devices list has the *QMidiArp* input activated. *QMidiArp* actually has two separate outputs that can be assigned to different arpeggiators (for splitting the bass and the lead sounds, for example) but for now it's easier to keep to the first one.

The next step is to make sure the intended track is connected to the synthesizer, by clicking on the channel name and selecting the corresponding output. This can generate feedback errors if it's connected to *QMidiArp* by mistake. The final thing to do is make sure that the metronome is disabled by clicking on its icon in the main Transport window (otherwise you would record the metronome ticking). Next, record-enable the track and press Record on the Transport control before playing something creative on the keyboard.

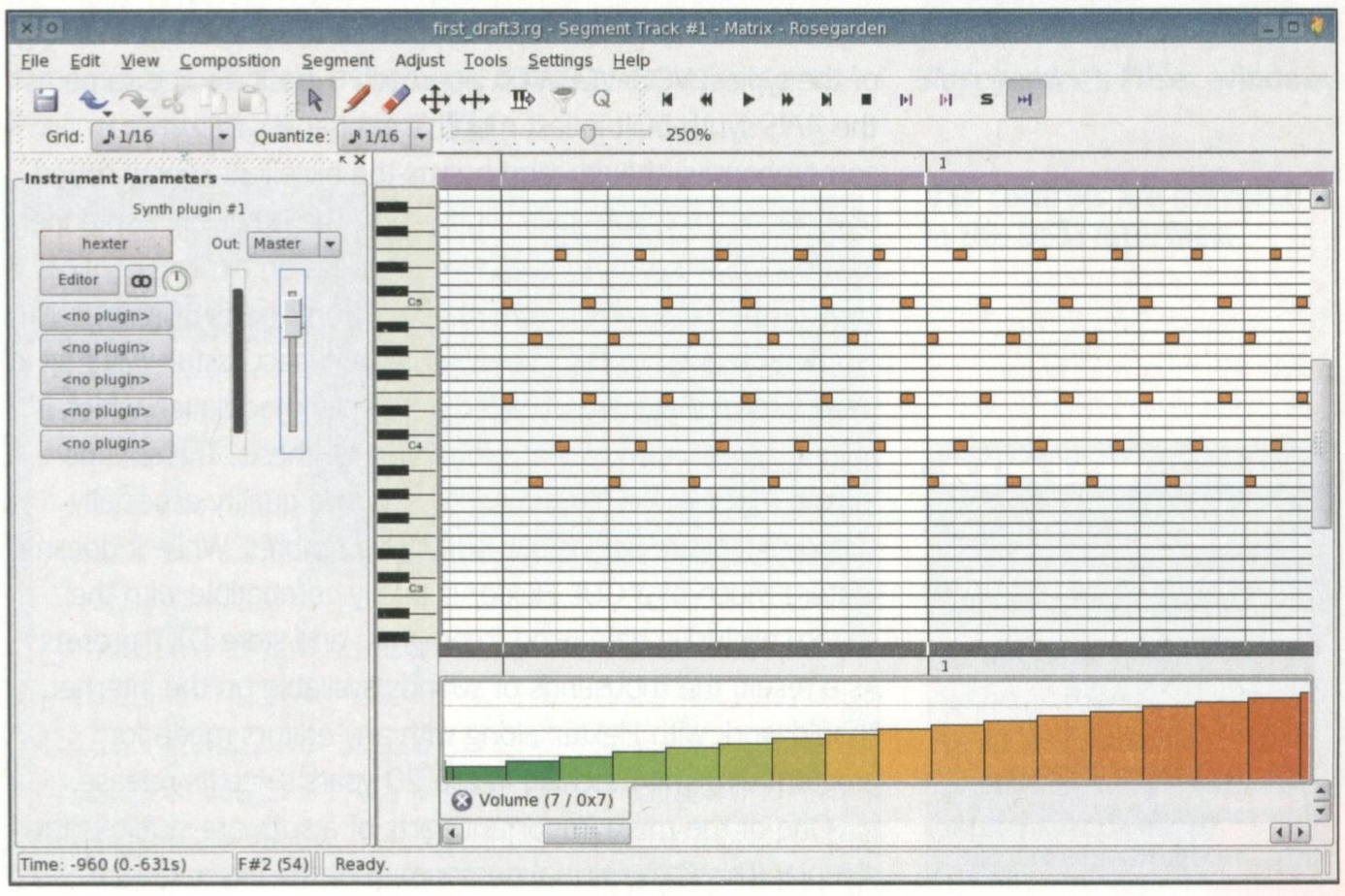
After the recording is stopped, the track should contain a single block full of notes. If there are spaces in the recording, they can be trimmed down to a more manageable size using the Split tool (F7), one of *Rosegarden*'s best features. Playing the track should produce a sequence identical to the recording, and as long as *QMidiArp* and *Rosegarden* shared the same tempo the notes should fit exactly into the bars and beats on the timeline. After opening the block, the notes won't fall exactly on to the time-grid, but this can be solved by using the Quantize tool. Quantization shifts notes on to the closest note division, and with the arpeggiator doubling the tempo twice to 16 beats in a bar, selecting 1/16 from the Quantize drop-down menu should move the notes to the correct places. To show the notes at their correct positions, the matrix editor's grid also needs to be changed to 1/16.

To make this block of notes more usable, you'll need to cut out the superfluous notes from the beginning and end before selecting everything and moving all the notes to the beginning of a bar. To make the sequence a little more versatile it's better to move the bass notes to another track so they can trigger a different sound. This can easily be done by selecting all the lower notes with the selection marquee and cutting them from the block. After this, close the matrix editor and draw a new block on to an adjacent track. Opening the matrix editor for this block and pasting the clipboard contents should insert the bass notes from the previous track.

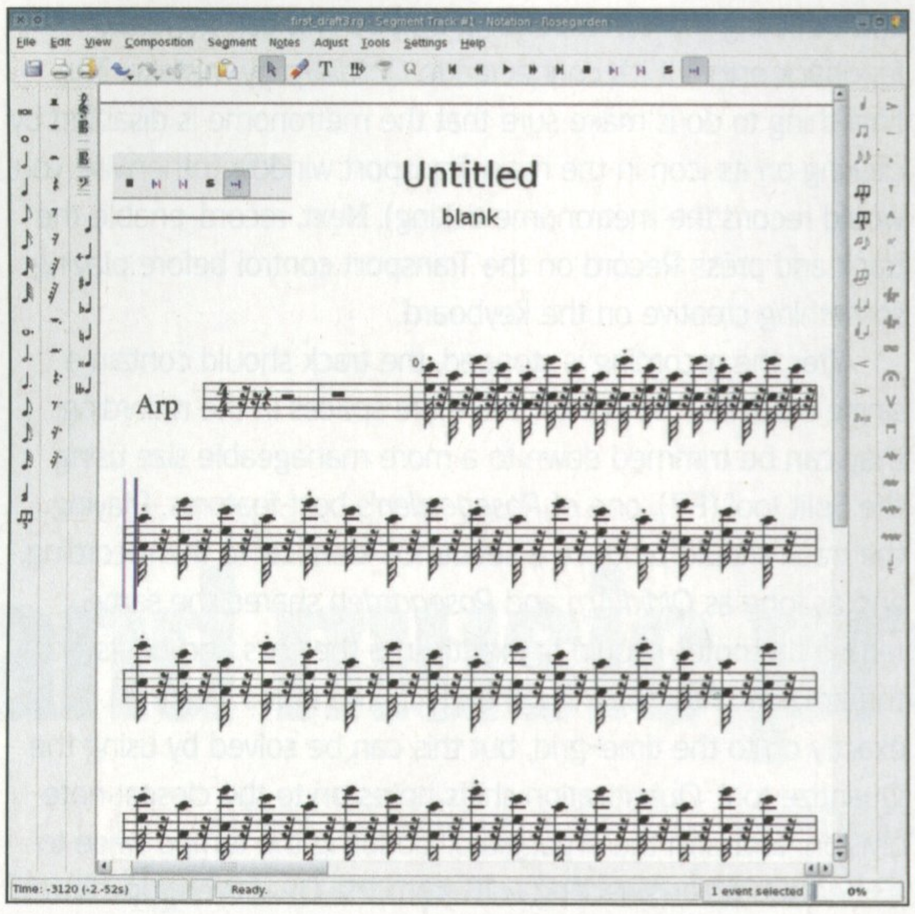
Virtual studio

As *Rosegarden* was developed to compete with similar applications available for other systems, one of the writers' main goals was to create an integrated audio environment. This means providing not only for both audio and MIDI tracks but also internal effects, synthesizers and mixer routing – all from a single application. While Linux doesn't share the same snappy acronyms as Windows or Mac with their VST and VSTi, the same functionality is becoming available with the emerging LADSPA (Linux Audio Developer's Simple Plugin API) and DSSI (Disposable Soft Synth Interface) standards, designed for »

The output in the matrix editor looks harder than it is.



Impress your friends with the notation editor.



plug-in effects and synthesizers respectively. The best way to work with software synthesizers in *Rosegarden* is to use its DSSI integration. It's important to note that *Rosegarden* needs to be configured to support these plugins, needing both the DSSI library and its header files installed and specified when compiling *Rosegarden* (using **--with-dssi** with the configure script). If the DSSI install has been successful, then choosing the output (right-click on the channel name) for a *Rosegarden* channel should include not only the MIDI and audio channels, but Synth Plugin devices as well.

DSSI is an API for audio plug-ins, aimed at bringing the power of VST instruments to Linux. So far, though, only a simple standalone host and *Rosegarden* support DSSI plug-ins, but it's building enough momentum for other apps to include support. The main reason for using DSSI over other options is integration. Compatible software is able to directly control DSSI through its own interface. This gives the impression that the DSSI is part of the same software, and means that projects containing DSSI plugins can be saved and restored without concern for loading any external synthesizers and their associated Jack connections. It's even courteous enough to remember patch configuration.

Yet another great advantage of DSSI is that the audio chain stays within a single program, making it much easier to manage and opening the way for chains of effects to be added on an integrated DSSI channel, as with *Rosegarden*.

The DSSI API comes with several example synthesizers, but the two most useful were developed independently of DSSI. *Xsynth* is more of a traditional approach to synthesis, consisting of the typical VCO-VCF-VCA approach – basically the same as the *AMS* synth built in last month's tutorial. It's still comprehensive though, and boasts the excellent overall sound of a classic analogue-style synthesizer. The other big contender for best DSSI-synth is called Hexter and is an almost identical copy of the best-selling Yamaha DX7 from the 1980s.

What made the DX7 so different, and successful, was that it used a form of synthesis called frequency modulation which is quite different from typical subtractive synthesis. The sounds have a much more distinct and percussive quality especially suited to electric pianos and soft string timbres. While it doesn't feature much of a GUI, Hexter is totally compatible with the system exclusive data used to program and store DX7 presets. As a result, the thousands of sounds available on the internet should work with Hexter, along with any editors made for programming the machine in the 20 years since its release.

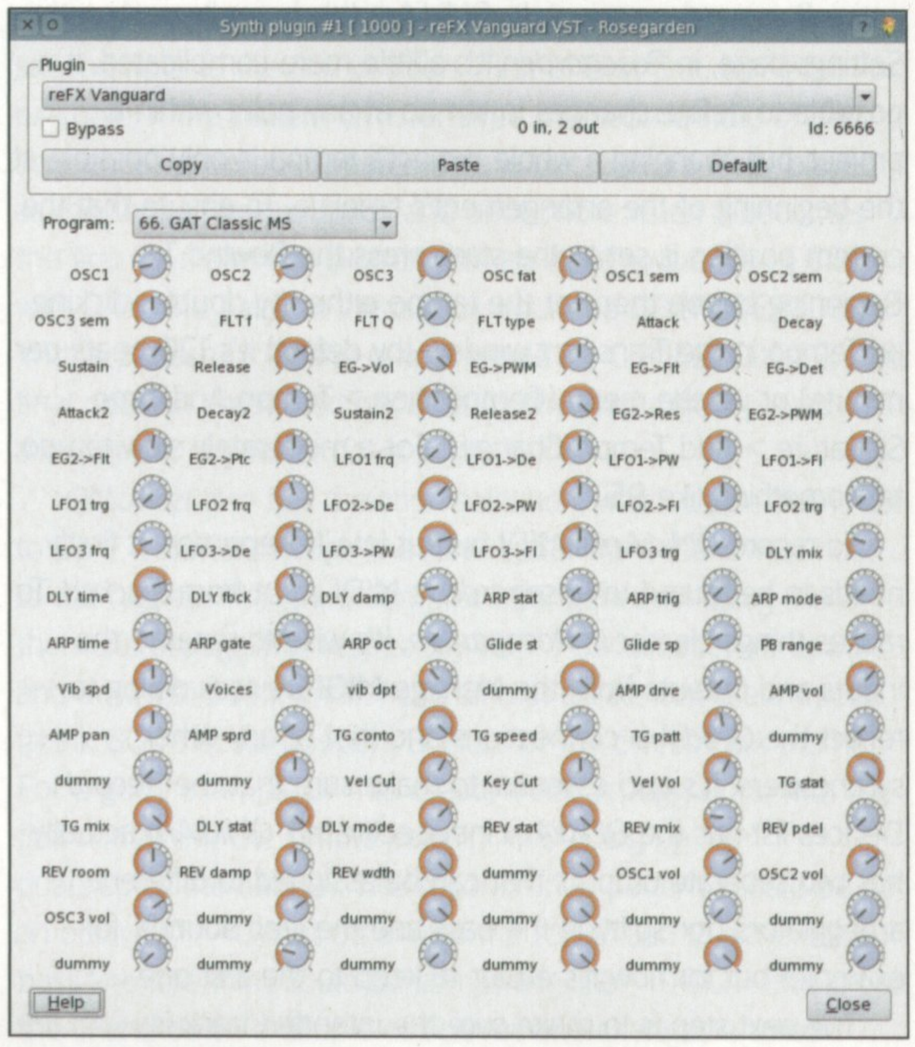
One of the more important parts of a software studio is the sampler. This is the virtual descendant of the old hardware responsible for many modern genres of music and many more that can't really be called music at all. While Linux doesn't have a native software sampler that can compete with the more serious applications for other platforms, there is a DSSI version of the popular FluidSynth, which is closely related to a sampler.

FluidSynth is a SoundFont synthesizer based on its version two specification, and is basically the sample-based equivalent of the synth that many current and older Creative soundcards feature. This means that the simple oscillators that are typically used in analogue-style synthesizers are replaced with audio samples. In all other ways the sample is treated the same as an analogue oscillator, in that it can be passed through envelopes and filters, and modulated with LFOs before finding its way to the audio output. These components are built into the software and hardware versions of each player, hopefully generating the same sound regardless of the platform.

Over the years, people have used cheap SoundFont hardware as a replacement for expensive samplers, and as a result there is a great deal of pre-prepared sound banks covering almost every genre of music. It's also perfectly possible to create custom sound banks using a SoundFont editor; the most comprehensive for Linux being *Swami*. The only downside to using a SoundFont synthesizer is usually the quality of its output. This isn't so much a result of the source material, but more the basic filter and LFO designs that can lack complexity compared with their retro counterparts. This isn't true for drum sounds, which seldom need to use these components, and therefore fit into the SoundFont framework perfectly. In fact, *Swami* makes a perfect editor for creating drum kits, as its graphical interface makes the usually arduous task of assigning drum sounds to keyboard notes relatively painless.

DSSI versus VST

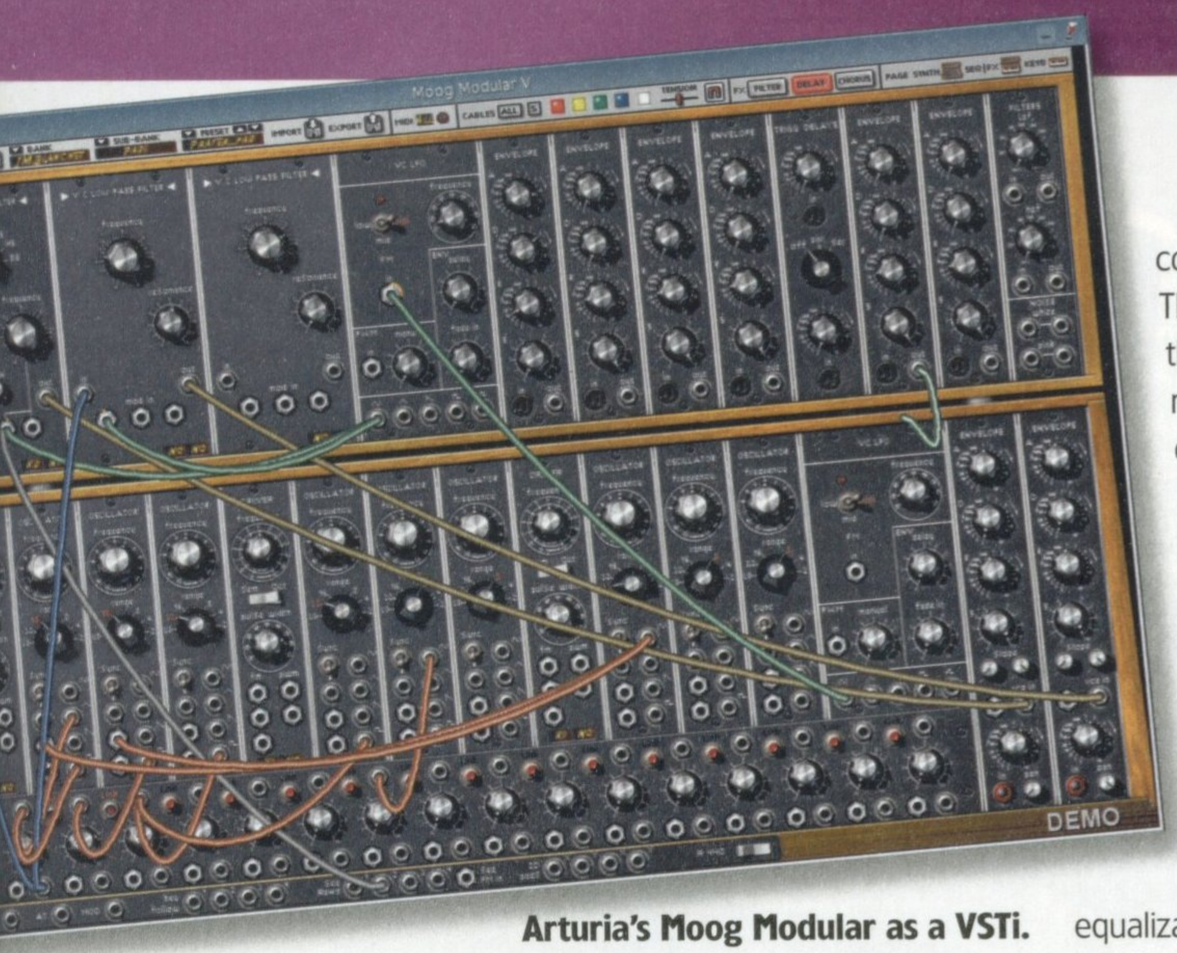
When it comes to DSSI, the area with the most potential in x86 architecture is VST (Virtual Studio Technology). VST is one of the primary technologies responsible for the incredible proliferation of software effects and instruments in recent years. Developed by Steinberg, the VST API basically builds a virtual studio protocol capable of replacing the wires and MIDI cables of a typical studio setup. Over the years, VST effects and instruments have grown to such an extent that they can now seriously compete with their hardware counterparts, and in



DSSI features its own synth interface.

LINKS
Software used this month

FluidSynth1.0.5
www.fluidsynth.org
DSSI 0.9
http://dssi.sourceforge.net
Rosegarden 0.9.91
http://rosegardenmusic.com



Arturia's Moog Modular as a VSTi.

many ways improve on their often cumbersome and unreliable designs. There are now VST versions of many great synthesizers of bygone ages, from Yamaha's CS-80 to the Moog Modular.

The trouble with VST is that it's only available for Windows and OS/X machines (with a brief sojourn to BeOS at one point); but some clever people have developed approaches to getting some level of VST functionality on x86 Linux using *Wine*. Initial attempts were focused on building a wrapper around the LADSPA Linux effects API. This was followed by a more successful effort, directly connecting to Jack, called *Jack_fst*. The DSSI API, however, is a much better container for VST instruments as well as effects, being purpose-built to provide a much more VST-like protocol than LADSPA (which was designed primarily for effects). While it's still in the early stages of development, DSSI-VST is proving to be very worthwhile.

Like Hexter and Xsynth, DSSI-VST is available from the DSSI repository and can be compiled against a recent version of *Wine*. Some synthesizers also require msvcp60.dll to be present in *Wine*'s Windows directory.

VST instruments and effects are usually supplied as DLLs (Windows libraries for the uninitiated). Most come with executable installers that can be successfully run from *Wine* before extracting its corresponding DLL from *Wine*'s installation and moving it to the VST_PATH location (usually /usr/lib/vst). After that, the new synth should become available to DSSI hosts as a DSSI client.

Part of the DSSI-VST package that isn't installed by default is called vsthost. This can either be executed directly or moved to /usr/bin and with a single argument for the VST DLL provides an independent audio and MIDI host for VST plug-ins. This enables it to work in the same way as *ZynAddSubFX*, for example, and is a versatile way of bringing VSTs to other apps.

Rosegarden is currently the only audio application to feature support for DSSI. Each track can be assigned to a Synth plug-in in the same way that it can be set to audio or MIDI, and the sound generated by the plug-in is routed back into *Rosegarden*. This makes it perfect for experimenting with the arpeggiator tracks recorded earlier. Sadly, one of the most powerful aspects of working with virtual instruments isn't yet implemented in *Rosegarden*, and that's automation. While it does support rather primitive editing of some of the more important controller data, such as volume and pan, it doesn't go the extra mile to provide the custom controlling and routing needed to be able to change virtual instrument parameters automatically.

While it often becomes essential to record the output from external software synthesizers (as with *AMS* last month), there's a slightly different approach to recording *Rosegarden*'s internal DSSI instruments, involving the mixer interface. Mixing consoles and audio production go hand in hand, and in studios worldwide the console is often the heart of the whole process. A mixing console's primary function is – obviously – to mix sound. This can be as basic as taking several inputs and mixing them down to a single output, or something altogether more complicated, involving multiple inputs with a variety of destinations. Modern consoles have become functional power-houses, and often impart much of their character into the overall sound. They function as part audio patch bay, part effects router and part parametric equalizer, and are usually directly linked to a recording device.

It's this same philosophy that's made the virtual mixer the centre of software audio, and it serves the same function as a console. Software mixers often provide effects channels, sub-grouping and equalization in the same way as their hardware counterparts. Routing and recording audio through the mixer is often called sub-mixing, and it's using this approach that makes recording the internal DSSI instruments possible.

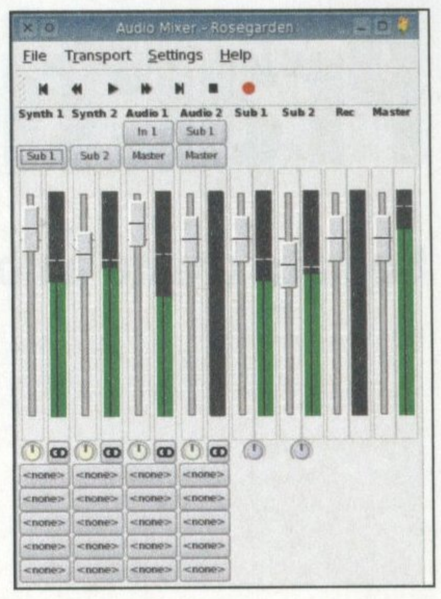
Mixing instructions

The first stage is to disconnect any external connections to *Rosegarden* from the *qjackctl* Connect window. These connections are usually direct from a soundcard's inputs, and while making the recording of external hardware possible, it can introduce unwanted noise into the audio signal path. Currently, *Rosegarden* doesn't support recording directly from the DSSI channels, and the way around this problem is to send the audio signal to a different bus. Busses are just buffers, and they're useful for all kinds of reasons. One example is with recording a drum kit where each sound is recorded to a different channel. Often, the whole kit needs to be treated as a single channel, and this is easily achieved by routing them all to a separate bus and controlling them as a single channel from there.

By default, *Rosegarden* doesn't have any additional busses, but they can be added from the audio mixer by selecting Settings > Number of Submasters > 2. In *Rosegarden*, they all output to the master bus (which is where the sound is output) and are added to the mixer window. To reroute one of the DSSI channels to a different bus simply left-click on the Output destination button just above the channel's fader.

The next stage is to set the input source to the new Sub bus, and this is done with the Input Record Source button above the Output–Destination button on an unused audio channel. This needs to be changed to Sub 1, which should be the output from the Sub 1 bus. After record-enabling the destination track, Sub 1 can be recorded by pressing the Record button; after which, the DSSI synth's output should be rendered to the track.

The final stages of music production are as important as the composition itself. Next month we'll cover various effects and the final mastering of a project. LXF



***Rosegarden*'s Mixer window.**

VST controls are mirrored in the DSSI interface.

NEXT MONTH

As this series draws to a close with next month's issue, we'll cover the final stages of audio production, using effects and mastering techniques.