From Polyphony To Digital Synths

Synth Secrets

- Synthesizers > Synthesis / Sound Design, Synth Secrets

By Gordon Reid

Published January 2001

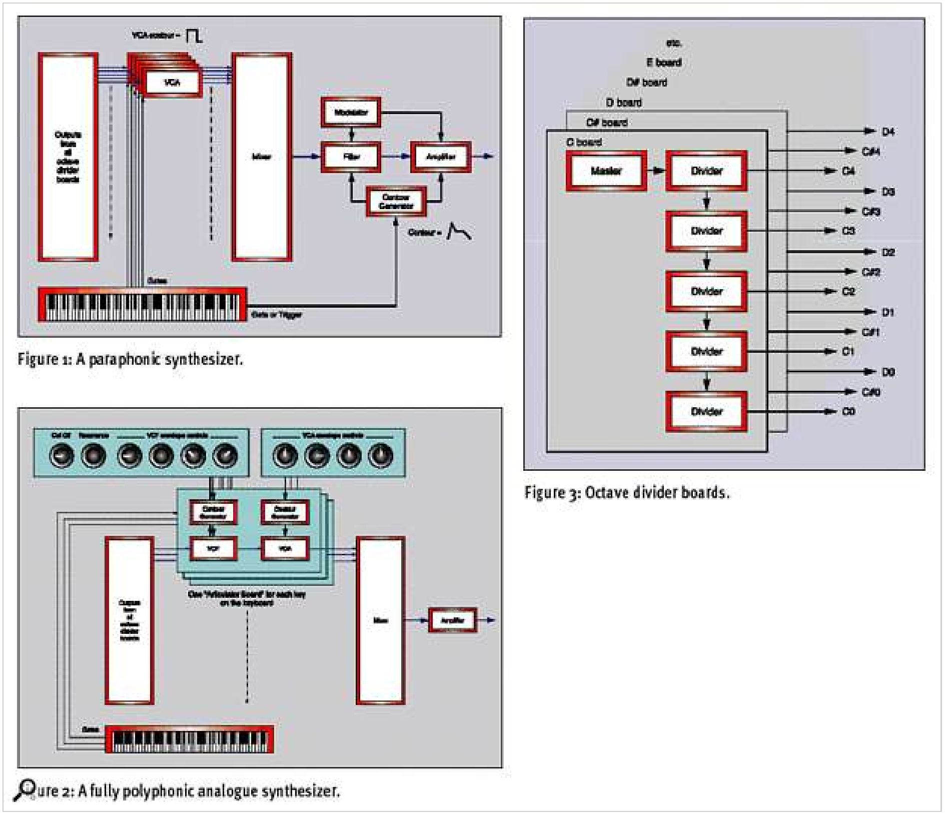


Figure 1: [top-left] A paraphonic synthesizer. Figure 2: [bottom-left] A fully polyphonic analogue synthesizer. Figure 3: [right] Octave divider boards.

This time, Gordon Reid explains how various analogue synth manufacturers attempted to create workable polyphonic synths by employing digital technology.

Last month, I showed that a truly polyphonic synth must be able to shape the tone and loudness of any sound independently of any other sounds or notes that it is producing at the same time. We then looked at one way in which an analogue synth can achieve this — ie. by providing filters, amplifiers and contour generators for every key on the keyboard. Unfortunately, an instrument of this design is expensive to manufacture, and its complex architecture usually results in unreliability. Moog provided total polyphony in this way on their Polymoog, and sure enough, this keyboard suffered from being too expensive and unreliable.

Unsurprisingly, most other manufacturers adopted different ways of circumnavigating this problem — and this month, I'll try and explain how exactly they did this. However, before considering the most common polyphonic architecture, I would like to look at one of the strangest analogue keyboards ever devised. It's not particularly obscure, nor very rare, but it has the weirdest architecture I've ever encountered. It's the Crumar Trilogy.

In this article...

In this Series

Crumar's Cut-down Paraphonic Approach

To understand the Trilogy fully, you'll need a quick reminder of the content of last month's Synth Secrets. Figure 1 (above) shows the architecture of a 'divide-down' paraphonic synth on which only the first note played benefits fully from the Attack and Decay stages of the contour generator, and only the last note benefits from the Release. Figure 2 (above) depicts a fully polyphonic 'divide-down' synthesizer — such as the Polymoog — that offers a VCF/VCA/EG 'articulator' board for every note on the keyboard.

As I've just recapped, the latter is expensive and potentially unreliable. A valid question is therefore: is it possible to devise an analogue synth that combines full polyphony and correct articulation of all notes, but with lower cost and greater reliability? Unfortunately, the answer to this is no, although the Crumar provides a unique and interesting compromise.

Let's go all the way back to octave-divider boards (see Figure 3, above). You'll remember from last month that each board uses a master oscillator related to a given note on the keyboard, and then 'divides down' that frequency to generate all the instances of that note across the keyboard.

Now let's take just one of these boards (the one that creates all the Cs, for example) and give it its own contour generator, filter and amplifier. The result is, in essence, a dedicated paraphonic architecture for all the Cs alone, and it looks like Figure 4 (below).

If we now stack 12 of these boards — one for each note in the octave-- we have a weird synth that articulates every note correctly, *provided* that you never play more than one instance of any given note. For example, if you play a C1/E1/G1 triad, each note will speak correctly whether you play them simultaneously, as a broken chord, as an arpeggio, or even within a solo. However, the moment that you add another 'C' to create the octave chord — C1/E1/G1/C2 — the second 'C' cannot be correctly articulated. This is because the appropriate contour generator, filter and amplifier have already received a Gate, and are some way along their ADSR response. I have shown this mythical synthesizer-- the 'GR1' — in Figure 5 (below).

The 'GR1' is mythical because, to the best of my knowledge, there has never been a commercial synthesizer that works in this way. The Crumar Trilogy, however, came close. If you ignore its organ and string ensemble sections (the basic sounds of which are tapped from the divide-down boards and passed through appropriate treatments to obtain the desired timbres), you find that its architecture looks very similar to the 'GR1'. However, there are two important differences.

Firstly, each set of notes has two independently tunable oscillators. The number of master oscillators and dividers is therefore double that shown in Figure 5 (see Figure 3).

The second difference is more radical. In what could only be a cost-cutting exercise, Crumar made the Trilogy not 12x paraphonic, but 6x paraphonic. This means that all the Cs and all the F#s share a contour generator, filter and amplifier, as do all the C#s and Gs, all the Ds and G#s... and so on up the octave (see Figure 6).

Unconventional though this architecture may be, it remains surprisingly usable. For example, on what other keyboard could you hold a drone — say, a low C — and play a solo above it which retriggers the drone every time that you play another C or an F#, but at no other time? Well... on no other. Add to that the Trilogy's ability to layer the strings and organ sounds derived from oscillator bank 1, and you have something quite unusual — not quite a true polysynth, but much more than a basic string machine or paraphonic synth.

Despite its curiosity value, the Trilogy possesses a structure which is clearly unsuitable for a fully polyphonic analogue synthesizer. Nevertheless, it demonstrates an important consideration in electronic design: if you use fewer components, your product will cost less and will be more reliable, all other things being equal.

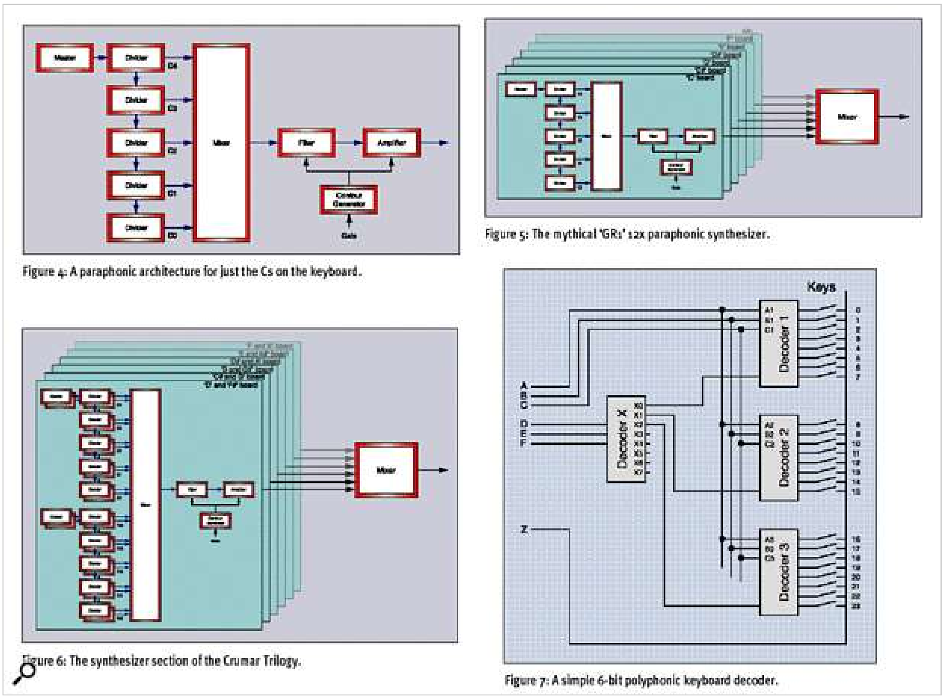


Figure 4: A paraphonic architecture for just the Cs on the keyboard.
Figure 5: The mythical 'GR1' 12x paraphonic synthesizer.
Figure 6: The synthesizer section of the Crumar Trilogy.
Figure 7: A simple 6-bit polyphonic keyboard decoder.
Master
Divider
Mixer
Filter
Amplifier
Contour Generator
Gate
Keys
Decoder 1
Decoder 2
Decoder 3
Decoder X

Figure 4: [top-left] A paraphonic architecture for just the C-notes on the keyboard. Figure 5: [top-right] The mythical 'GR1' 12x paraphonic synthesizer. Figure 6: [bottom-left} The synthesizer section of the Crumar Trilogy. Figure 7: A simple 6-bit polyphonic keyboard decoder.

Oberheim & The SEM

So let's look at polyphony a different way — the way Oberheim chose to in the mid-'70s. Remember, for every note played on a fully polyphonic instrument, you need a dedicated sound source, dedicated filters, dedicated amplifiers, and dedicated contour generators. So how about cutting down on components, by reducing the amount of polyphony and then assigning a complete synthesizer 'voice' to each note as you play it? This seems like a good idea, but it raises problems of its own. In particular, if we're going to limit the polyphony to just a handful of notes... which handful do we choose? Of course, the answer to this has to be 'whichever ones you happen to be pressing at any given moment' — otherwise, the synth is unplayable!

The first affordable synth that allocated voices to notes was the Oberheim 4-Voice, and we're going to see how it did this. But before going any further, here's a health warning for analogue anoraks: for the first time in Synth Secrets, we're going to take a detour into some real digital electronics. This is unavoidable, because logic circuits are the only practical and economical way to determine which keys are depressed at any given moment on a polyphonic keyboard (this also explains why there were no polyphonic synthesizers before 1974... the necessary technology did not exist!).

Binary Numbers & Keyboard Scanning

I'll start by refreshing your knowledge of binary numbers. As you know, binary is simply a number system, just like the decimal system used in everyday life. However, instead of counting from zero to nine before adding a new column, as in decimal, we only count from zero to one before adding a new column. Consequently, whereas the decimal system counts 0, 1, 2, 3, 4, and so on, binary represents the same numbers as 0, 1, 10, 11, 100... and so on. I have shown the equivalence of some decimal and binary numbers in Table 1,

INPUT D	INPUT E	INPUT F	DECIMAL RESULT
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Table 1.

As you can see, every time a number in binary is depicted using all 1s, we add another column of digits on the left and start again. This should be familiar in principle, as we do the same in

Readers' Ads

VIEW ALL ADS CREATE FREE AD

On the same subject

- The Secret Of The Big Red Button
July 2004
- More Creative Synthesis With Delays
June 2004
- Creative Synthesis With Delays
May 2004
- From Analogue To Digital Effects
April 2004
- Synthesizing The Rest Of The Hammond Organ: Part 2
March 2004

From the same manufacturer

- Vintage Vice!
February 2021
- Sequential Prophet-5 & Prophet-10
February 2021
- Dave Smith of Sequential
May 2019
- Sequential Prophet X
November 2018
- Sequential's Prophet 2000 Samplers
January 2004

SIGN UP TO
SOS NEWSLETTERS

Latest SOS Videos

decimal counting — when we reach 9, 99 or 999, we add a new column and start again at 10, 100 or 1000.

In digital electronics, people don't talk in terms of 'digits', but instead of 'bits'. Moreover, for any given number of bits we add all the leading zeroes to the binary number. Therefore, taking a decimal number from Table 1 — say, six — and writing it as an eight-bit binary value, we would write this not as 110, but as 00000110.

The 6-Bit Keyboard Decoder

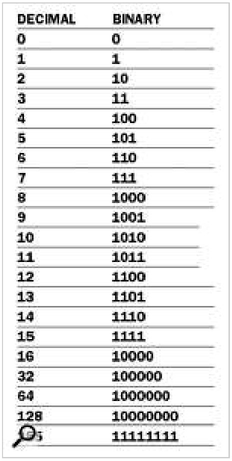
As we've discussed, binary numbers deal only with zeros and ones. But take a mental leap to the left, and you can just as easily say that binary numbers deal only with either/or states. These could be 'high' and 'low' voltages, or 'open' and 'closed' or 'on' and 'off' switch positions. And (ignoring for now such complications as velocity sensitive and pressure sensitivity) a keyboard is merely a rack of well-engineered on/off switches. So, here's another question: is it possible to devise a method that expresses the on/off status of a bunch of keys by representing them as a series of zeros and ones?

The answer lies in Figure 7 (see earlier), which represents the 'digitally scanned polyphonic keyboard' developed in 1973 by Dave Rossum and Scott Wedge of Eμ (later renamed Emu) Systems.

If you've never studied digital electronics, you might think that you can't possibly understand this diagram, but I assure you that you can. Stick with me for a while longer, and I'll demonstrate how the 24 switches on the far right of Figure 7 can be used as the key contacts of a polyphonic synthesizer. Furthermore, I will perform magic, and show you that the single data line Z can carry all the information needed to tell the instrument which combination of keys is pressed at any given time.

But before moving on, we've got to discard all our previous understanding of monophonic and duophonic keyboard mechanisms. On most of these, the action of depressing a key closes a set of contacts that determine a pitch CV, as well as firing off a Gate and/or Trigger pulse. In contrast, a polyphonic keyboard scanning system must — by its very nature — relegate the keys to a passive role. The active element in the system is the digital (ie. binary) scanning circuit that recognises whether you have pressed any keys or not.

Now let's return to Figure 7 (above). We'll start by considering what happens in Decoder X if you treat the three inputs D, E and F as a binary number. Let's assume that, if the voltage at the input is 'high' (+5V) it represents a 1 and if it is 'low' (0V) it represents a 0. Since there are three lines, we can use them to represent a 3-bit binary number. Table 2 shows all the possible values.



DECIMAL	BINARY
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
32	100000
64	1000000
128	10000000
[illegible]	11111111

Table 2.

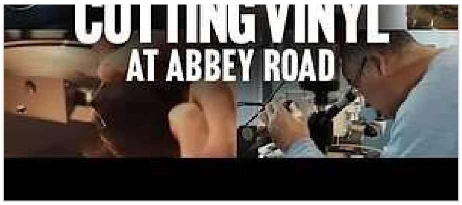
Clearly, the three bits give a maximum of eight possibilities (including zero) so the three inputs D, E and F can determine which of (up to) eight outputs from Decoder X can be set 'high' at any given moment.

Now let's look at the lines A, B, and C. These are routed to each of the subsidiary decoders, Decoder 1, Decoder 2, and Decoder 3. Again, they can represent the eight numbers from zero to seven. Therefore, Decoder 1 sets its '0' output high when it receives '000' from A, B and C. It sets output '1' high when it receives '001' from A, B, and C... and so on.

The extra element is the line X0 that leads from Decoder X to Decoder 1. This is an 'enable' line, and it ensures that Decoder 1 will only do its stuff when X0 is 'high'. Likewise, Decoder 2 is only active when X1 is 'high', and so on.

Still with me? If so, let's now analyse what happens as we count in binary across the six data lines, D, E, F, A, B, and C:

- With DEF = 000, Decoder 1 is enabled, and A, B and C count from 0 to 7, in turn setting the voltage 'high' on the input to each of the switches 0 to 7.
- When the count reaches 000111, it continues from 001000.
- With DEF = 001, Decoder 2 is enabled, and A, B and C count from 0 to 7, in turn setting the voltage 'high' on the input to each of the switches 8 to 15.
- When the count reaches 001111, it continues from 010000.



Cutting Vinyl At Abbey Road

4 days 23 hours ago.

Your Questions Answered

2 months 1 week ago.

Radiophonic Recording At Eve Studios

3 months 6 days ago.

- With DEF = 010, Decoder 3 is enabled, and A, B and C count from 0 to 7, in turn setting the voltage 'high' on the input to each of the switches 16 to 23.
- When the count reaches 010111, the system resets to 000000 and the cycle begins anew. Mind you, if I had enough paper, I could count up to 111111, use eight subsidiary decoders, and have a maximum of 64 switches!

Now for the final piece of the jigsaw. Let's say that, for example, the input data DEFABC is 001001. This means that Decoder X enables Decoder 2, and that key 9 is 'high'. The important thing here is that, across the whole keyboard (yes... those switches in Figure 7 represent real keys on a synth keyboard) only key 9 is 'high'. But if you don't have key 9 depressed at that moment, the switch is open, and the voltage on the Z line (the output) remains 'low'. If, on the other hand, the key is depressed, the voltage on Z is 'high', and the system detects the status of the pressed key.

This result is beautifully simple and elegant. Simply by counting in binary from 000000 to 111111 we can cycle through all the keys on (up to) a 64-note keyboard, detecting in turn whether each key is depressed or not. The digital electronics in the synthesizer monitors line Z, and knows that a 'high' value at any given instant in the scanning cycle represents a specific pressed key. And, if we want to scan a wider keyboard, we just add more bits to the address lines. It's brilliant!

Phew! Now Back To '70s Oberheims...

All the Oberheims sold from 1974 to 1977 were based upon monophonic modules called SEMs (Synthesizer Expander Modules). Each SEM offered two oscillators, a multi-mode filter, an LFO, two contour generators, and an amplifier (see Figure 8, below).

The 4-Voice had, as you may have guessed, four of these, each with CV and Gate inputs. This would suggest that, thanks to the digital scanning technique, it was possible to access each of the SEMs from the keyboard, as shown in Figure 9 (below).

Of course, it's one thing to be able to say which keys are depressed at any given moment, but that's a far cry from generating the CVs, Gates and Triggers that tell a bunch of SEMs which notes to play. Figure 9 must be missing something...

The missing element is the so-called Voice Allocation Unit. You'll be pleased to know that the circuitry of this is too complex to discuss here, but the principle of its action is simple enough: it takes the note information generated by the decoder, checks to see whether there are any unused SEMs available, and then generates and sends the CV and Gate information to those SEMs. Of course, this isn't the end of the story. Matters are complicated considerably by a Split mode that divides the keyboard into two virtual keyboards, each with a predetermined number of SEMs allocated to them. Then there is Unison. This layers the voices on top of each other, reducing polyphony to just one note (ie. monophony).

But, ignoring these additional complications, let's consider the result of playing more notes than the synthesizer has voices. In the case of a 4-Voice, what happens when you press five notes simultaneously?

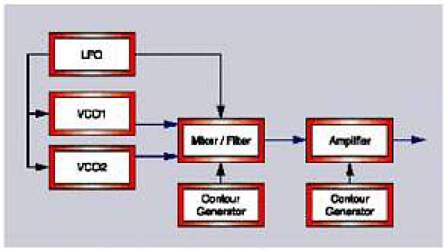


Figure 8: The basic modules within an Oberheim SEM.

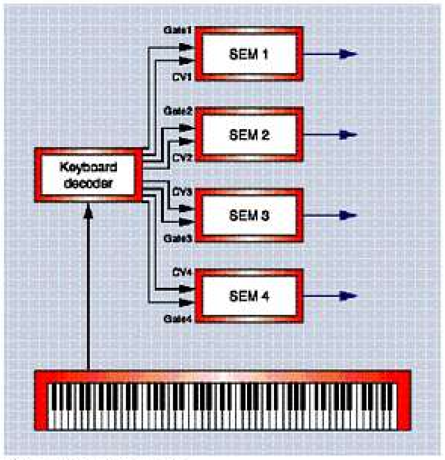


Figure 9: A 4-voice synthesizer?

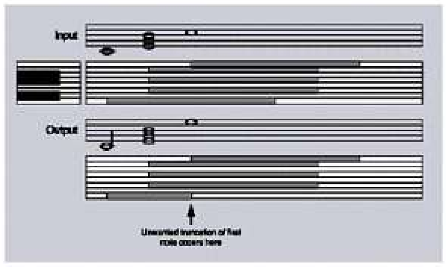


Figure 10: Note stealing on a 4-voice polysynth.

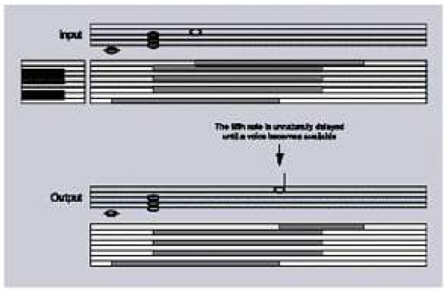

Input
The fifth note is unnaturally delayed until a voice becomes available
Output

Figure 11: Delayed notes on a first-note-priority 4-voice polysynth.

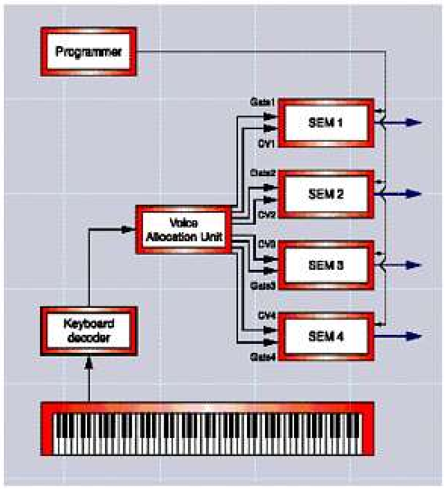


Figure 12: The basis of the Oberheim 4-Voice, showing the Voice Allocation Unit and programmer.

Figure 8: [top-left] The basic modules within an Oberheim SEM. Figure 9: [top-right] A 4-voice synthesizer? Figure 10: [left-middle] Note-stealing on a 4-voice polysynth. Figure 11: [left-bottom] Delayed notes on a first-note-priority 4-voice polysynth. Figure 12: [right-bottom] The basis of the Oberheim 4-Voice, showing the Voice Allocation Unit and programmer.

The answer to this depends upon the way in which the Voice Allocation Unit allocates the SEMs. For example, the 4-Voice offers one option in which it will cycle through each voice in turn. If you press just one note at a time, this means that the SEMs are allocated 1-2-3-4-1-2-3-4... and so on, as you play. But when you hold the first four notes, this system leads to 'note stealing' if you press a fifth key simultaneously (see Figure 10 above).

One way round this is to give earlier notes priority over later ones. This is the 'first-note priority' system discussed in Part 18 of this series (see SOS October 2000) but now applied polyphonically. Unfortunately, this will delay the onset of later notes, and the results may be even less desirable than note stealing (see Figure 11 above).

As you can see from Figure 9 (above), the Oberheim 4-Voice was, in essence, four independent monosynths screwed into a wooden case together with a digitally scanned keyboard and its associated electronics. This meant that you had to set up all of the SEMs identically to play it as a conventional polyphonic instrument. Given the vagaries of this vintage of analogue electronics, this was all but impossible. SEMs are not famous for their stability, and getting four of them to stay in tune, correctly scaled, let alone with identical filter and contour characteristics... well, you could forget that! It's small wonder that few players took advantage of the synth's polyphonic potential. Indeed, some players just programmed each of the SEMs individually, and — in Unison mode — played the 4-Voice as one of the most overpowering monosynths of all time.

A year after its initial release in 1974, Oberheim added a 16-memory programmer to the 4-Voice, which should have solved the first problem (ie. setting the individual SEMs to the same sound for polyphonic use). Unfortunately, the programmer was unable to store and recall all of the parameters relating to a patch. Most seriously, it could not remember the resonance or the filter type selected.

The basic structure of the 4-Voice can be seen in Figure 12 (above). Amazingly, the keyboard scanning and voice allocation functions are carried out entirely in hardware, with logic gates determining which keys have been pressed, which SEMs are available, and how the two should be paired together. In 1976, Emu picked up the baton of progress, took a huge leap for synthesis and adopted a microprocessor as the 'brain' of their 4060 keyboard — the same microprocessor that, in 1977, went on to become the basis for Sequential's Prophet 5.

But why stop there? If you'd worked out a mechanism for scanning the voltage status of a bunch of key switches ('high' or 'low') surely you could use the same mechanism to determine the states of front-panel switches such as voice selectors? Sure you could! For that matter, is there any reason why you couldn't then use it to measure voltages other than 0V or +5V? To put it another way, could you not use it to measure the settings of the knobs and faders on the synthesizer's control panel? You certainly could. Not only that but, using a suitable A-D converter, you could translate these values into a digital format suitable for storing in computer memory. And of course, people did. Blimey! This month, I've not only stumbled upon the secrets of practical polyphony, but also the way to design analogue synthesizers with memories.

So consider this... If a polyphonic synth allocates a limited number of voices to the notes as you play, it contains a significant amount of digital circuitry. Likewise, if a polyphonic synth has switches that select between preset patches, it contains a significant amount of digital circuitry. Similarly, if a polyphonic synth has user-programmable memories, it too contains a significant amount of digital circuitry. Therefore, a totally analogue polysynth can only be one that offers total polyphony provided by independent articulators for each note, and which has no presets and no memories. So here's this month's synth secret:

In mainstream synthesis, there have only ever been two totally analogue polysynths: the Korg PS3100 and PS3300.

A Real Example

To finish this month's article, let's look at a real synth — the Sequential Circuits Prophet 600 — to illustrate all the above.

Released in 1983, this has six analogue voices that — unlike SEMs — are carefully calibrated to sound as close to identical as possible. However, the 600 uses a Z80 microprocessor to scan the keyboard, scan the control panel, and manage its 100 memories. The Z80 is even pressed into sound generation (as opposed to sound control) duties, calculating software-generated contours and LFOs that are applied to the synth's analogue VCOs, VCFs and VCAs. It all works like this...

Every 200th of a second the Z80 calculates the values of the envelope generators, the LFOs and the effect (if any) of glide. It then refreshes the LEDs on the top panel, looks at either the pitch-bend or modulation wheel, refreshes a number of the internal control voltages, and then checks one (and only one) of the control panel knobs. Next, it scans the keyboard and, if you've played a note during that period, works out the voice assignments. This means that — depending upon the exact moment at which you press a key — your playing may be delayed by up to 5 milliseconds. You can also detect the consequences of the scanning and calculating when you adjust the Prophet 600's control knobs by small amounts. Listen carefully, and you can hear the effect as the processor jumps from value to value. This is one source of the famous 'digital zipper noise'.

Nowadays, keyboard scanning, note allocation, and numerous other functions are controlled by firmware within special chips called ASICs — Application Specific Integrated Circuits. You can think of these as dedicated microprocessors that have been pre-programmed to perform specific tasks). This is as true for the modern generation of analogue polysynths — for example, the Studio Electronics Omega 8 and the forthcoming Alesis Andromeda — as it is for the plethora of virtual analogue (VA) synths and digital workstation keyboards.

Thanks to Dave Smith, ex-head of Sequential Circuits, for supplying the photograph of the Sequential Prophet 600.

Random Voice Assignment

If you open a vintage polysynth such as an Oberheim OB8, you'll see on each voice board an LED that tells you whether this voice is playing or not. You can then press a succession of keys and see the LEDs march across the synth from 1 to 6 (or 1 to 8) before starting back at voice 1 again. If you hold a four-note chord, you can see, say, voices 1 to 4 light up, and then voices 5 to 8 (or whatever) cycle as before.

However, on a couple of early polysynths, the voices do not always rotate in such a strict, cyclic fashion, and this offers an unexpected benefit. Each of the voices in an analogue instrument will sound slightly different from the others — maybe with a different amount of detune, or with filters that are slightly more open or closed. These differences, if they are not too extreme, are a major source of the so-called 'organic' warmth of vintage polysynths.

Nevertheless, if the voices always play in the same order, you may occasionally hear a disturbing consistency as you perform, especially if you're playing a solo line. Let's suppose that voice three of a six-voice synth is slightly more 'open' than the others. If the voices speak in strict rotation, you'll hear your solo doing something like this: 'do-do-dee-do-do-do... do-do-dee-do-do-do...' This will place your performance firmly within electronic territory.

But if the synth's voices do not cycle in a predictable fashion, the same line may go: 'do-dee-do-do-dee-do... do-do-do-do-do-dee...' which will be much closer to the natural variations of tone and tuning of a 'real' musical performance.

Nowadays, of course, digital synths have 'analogue feel' parameters that add small random fluctuations to the sound, giving rise to much the same effect.

Find ALL Synth Secrets Parts

Previous article Next article

blinddrew
Recording: Gear & Techniques 28 Mar 2022, 10:18

Spotify news

- Self-Promotion
- Feedback

Contact Us Cookie Policy Help Privacy Policy Terms of Use