AN ALIVE SQ80-M | Probably more seldom than a Sasquatch ...

From rumors to reality: The Ensoniq SQ80-m

By: Rainer Buchty

Often enough, synthesizer companies make odd decisions. Upgrading one member of a synth family, but not the other, as in the case of ESQ-1/ESQ-m is one of them. It wouldn't have taken them much to also provide a module version of the SQ-80. In fact, the changes to hardware and software are so small, that anyone who knows their solder iron's hot end from the cold one can turn their ESQ-m into an SQ-80m.

Wait, what?

Ensoniq users know about the similarities and differences between the ESQ-1 and the SQ-80. Sure, there's the polypressure keyboard, a disk drive, and 64kB of sequencer memory. But when it comes to synthesis, the SQ-80 is essentially an ESQ-1 with quadrupled waveform memory, offering 75 instead of 32 waveforms. These not only include additional single-cycle waves but also multi-cycle ones. It therefore should be enough to apply some circuitry to the soundchip, the famous Ensoniq DOC (or Q-Chip, as Mirage users like to call it), upgrade the wave ROM, and apply some gentle changes to the synthesizer's operating system so that it can use the new waveforms. But is it really that easy?

The SQ-80 is a hack!

Mirage programmers know it by heart: the DOC can address a waveform memory of 128kB at maximum. And this is already featuring in the DOC's own *bank select* pin, switching between 64kB banks. That surely does sound familiar to ESQ-1 and Apple II GS users who only have access to such a single bank of waveform memory. The SQ-80, however, offers 256kB of waveforms – how does it do that? To answer that question, we need to look closer at those machines and how they use the DOC. By design, the DOC supports up to 16 audio channels onto which each oscillator can be routed. For this, it features four distinct channel address bits – for those not remembering calculus: $2^4 = 16$ – together with a channel address strobe, signaling validity of said channel address.

Given that the DOC supports up to 32 digital oscillators, it becomes somewhat obvious that when designing their sound chip, Ensoniq engineers had a 2-osc/16 voice synthesizer in mind. Truly powerful by the time of design, especially factoring in the built-in synchronization and amplitude-modulation capabilities – but unfortunately also with a glitch: for whatever reason, the engineers decided that only odd oscillators can sync even ones, but only even ones may amplitude-modulate odd ones. Depending on whether you want to use sync or AM, a voice now either occupies oscillators 0+1, 2+3, 3+4, etc. or 1+2, 3+4, 5+6, etc. – and suddenly your pair of oscillators becomes a triplet or just a sheer nightmare of oscillator-to-voice allocation. But three oscillators per voice, that does sound familiar, doesn't it? Exactly. It's what the ESQ-1 and SQ-80 use, and if you are brave enough to wade through assembly language, you will notice how this greatly eases oscillator allocation as you now can easily use sync and AM while always staying within a group of three oscillators.

But, wait, didn't we want to go for 16 voices? Three oscillators times 16 voices makes 48 oscillators in total, and the DOC only has 32! Right you are. In theory, the DOC could provide 10 voices per three oscillators in total, but as analog postprocessing (filters!) is expensive – and computer engineers love powers of two for obvious reasons – the machines only support 8 voices,

requiring only 8 audio channels for oscillator routing.

Now, it would be shame to waste that perfectly fine channel address bit, right? Since a programmer has full control of that bit and given that it directly corresponds with a currently played wave, it makes a perfect hack for waveform memory expansion: using this bit gives the SQ-80 the ability to select between two 128kB banks of waveform memory, and therefore providing a total of 256kB of waves.

I NEED YOUR BOOTS, YOUR CLOTHES, AND YOUR MOTORBIKE.

When the Terminator entered 1991, he was demanding three things from his first unlucky victims. Making an ESQ-1 or ESQ-m address 256kB of waveform memory takes only slightly more:

- a 74LS74-type flipflop
- a 27C020-type EPROM
- a 27C256-type EPROM
- an EPROM programmer

The 74LS74 contains two so-called D-type Flip-flops. These are essentially 1-Bit memories featuring a data input *D*, a clock input telling the 1-bit memory when to store data available at its input, and two outputs *Q* and $\bar{Q}$ where the first represents the currently stored value and the latter the inverted stored value. This guy we will use for turning the fourth (and in the ESQ-1/ESQ-m unused) channel address bit, into a proper waveform address bit.

By the time, Ensoniq was creating the ESQ-1, the biggest EPROM size available on the market for decent money was the 27C256, featuring 32kB of non-volatile storage. They could have used a 27C512, offering the 64kB they needed, but at design time it probably was prohibitively expensive compared to the extra board space needed. In 2014, we have vast amounts of memory at our hands, therefore we use a 27C020-type EPROM which gives us a whole 256kB as required.

What else do we need? Of course a new operating system so that the ESQ-m will make use of those extra waves. For this, we use a 27C256-type EPROM. In fact, if you are brave enough you could either recycle a wave EPROM or the ESQ-m's original OS ROM, assuming that either of those is an EPROM in your machine and not a ROM. (If it has a sticker on it and below that sticker sits some tiny window through which you can see a chip, then it's an EPROM. Now all you need is an eraser to wipe its previous contents and reset it to blank state.)

PROGRAMMING THE EPROMS

Let's start with the easy part. Grab the binary image of the patched operating system and program it into a 27C256 using your EPROM programmer of choice. Don't forget to verify the contents to be sure that programming went ok. Put a label on it which marks it as *SQ80-m OS1.3* (it essentially is a patched ESQ-m OS 1.2). Now grab the binary image of the 256kB SQ80 waveform memory, program it into the 27C020, and verify it. Label this *SQ80m WaveROM*.

Now replace the ESQ-m's OS ROM with your freshly programmed one and power the machine up. It should greet you already with the SQ-80m welcome screen. If not, make sure you have put in the OS correctly. That was easy, huh?

WIRES, WIRES, EVERYWHERE

You guessed it. Now comes the hard part. I always get sweaty hands when working on machines which belong to me (funnily never when I work on others' machines), and surely the following procedure should only be performed by experienced electronic technicians.

First, we need to wire that 74LS74 to the soundchip. In your machine, it will typically be marked *5503 DOC* with a second line reading *ICS1261D*. That's the one. Since the 74LS74 is to stay, you might first glue it back-to-back to the DOC, then start the wiring. Remember: If the IC sits on its pins, then the topmost pin left of the notch is pin 1. From this you count down, wrap to the other side, and count up again, so that the topmost pin right of the notch is pin 14 (in the case of 74LS74) or pin 40 (in the case of the DOC). If you go for the back-to-back glueing, also keep the pin orientation straight – as you turned the chip upside down, pin 1 is now the topmost pin *right* of the notch ...

In the below table you see the wiring of the 74LS74. Connect pins 2, 3, 7, and 14 to the DOC as listed. Furthermore, connect pins 1, 4, 10, and 10-13 to VCC which you now can pick from pin 14. Leave pin 5 open for the moment, we will need it later when wiring the Wave ROM.

At this stage you may want to power up the machine again. Play a few notes via MIDI. Does it sound as usual? Great. Then you did all the DOC wiring correctly. If the machine doesn't come up at all or doesn't play the notes correctly, switch it off immediately and check for wiring errors.

BRAIIIIINS!

Time to install the new waveform memory. Remove the two existing Wave ROMs (if they are EPROMs, you may want to recycle them). You will notice that the 27C020 has 32 pins, not just 28 as the original ones. Furthermore, as it's only one, not two, we need to apply some further changes. To ease later wiring (and avoid malfunction) bend up pins 22, 24, and 30-32. These are all on the chip's right side if oriented notch up. You also want to bend up pins 1-3 on the topmost left side. Now place the EPROM into either of the WaveROM sockets (it doesn't matter which one) so that it aligns with the *bottom* of the socket. Yes, that will place the two topmost pins on each side outside the socket – that's ok!

Wire the ROM according to the table on the next page. Firstly, being an EPROM, the chip has two control signals which are only required for programming, therefore we connect it to GND (VPP, pin 1) and VCC ($\overline{\text{PGM}}$, pin 31) respectively in order to deactivate them.

Furthermore, the chip – sitting "off socket" – doesn't make contact with +5V anymore, so this must be wired as well. But what's that magic 74LS373? It's a chip that sitting on the ESQ-m main board which the DOC uses for accessing waveform memory, therefore you will spot it sitting close to the sound chip. It will carry VCC on pin 20.

While you are at it, also wire its pin 19 to the Wave ROM's pin 3. Now, as the final step, connect the remaining pins to the DOC so that bank select,

74LS74		DOC		
Pin	Signal	Pin	Signal	Description
2	D1	10	CA3	Channel address bit 3
3	Clk1	6	CHSTB	Channel address strobe
5	Q1		n/a	Wave ROM address bit 17, see text
6	$\overline{\text{Q1}}$		n/a	unused, leave open
7	GND	19	GND	ground power supply
8	Q2		n/a	unused, leave open
9	$\overline{\text{Q2}}$		n/a	unused, leave open
14	VCC	39	VCC	+5V power supply
1, 4, 10-13	VCC		n/a	see text

WIRING OF THE 74LS74 DUAL D-TYPE FLIP-FLOP | (see text)

27C020			wire to		
Pin	**Signal**	**Chip**	**Pin**	**Signal**	**Description**
1	VPP	27C020	16	GND	power supply, ground
2	A16	DOC	40	BS	DOC Bank Select
3	A15	73LS373	19	SA15	Sample Address 15, see text
22	$\overline{\text{CS}}$	DOC	5	E	Wave-ROM chip select
23	$\overline{\text{OE}}$	DOC	3	$\overline{\text{CAS}}$	Wave-ROM output enable
30	A17	74LS74	5	$\overline{\text{Q1}}$	Wave-ROM address bit 17
31	$\overline{\text{PGM}}$	27C020	32	VCC	power supply, +5V
32	VCC	74LS373	20	VCC	power supply, +5V

WIRING OF THE WAVE ROM | (see text)

(DOC pin 40), Wave-ROM chip select (DOC, pin 5), and Wave-ROM output enable (DOC, pin 23) are wired to the corresponding pins of the Wave ROM. Sweaty hands? Told ya!

Now power on the machine for a last test. If it does not power on normally, switch it off immediately and search for any errors made. If it does power on, play some notes. Everything fine? Great! Now go for downloading some SQ-80 sounds or create some directly at the machine!

Some final words

So far, only little experience with SQ-80m OS 1.3 exists. Besides waveforms, there are also other – minor – differences between the ESQ-1 and SQ-80 sound engines, such as the second-release (for reverb-like effects) and changes in velocity scaling. Therefore, it might be that SQ-80 patches do not sound 100% accurate when played on the SQ-80m.

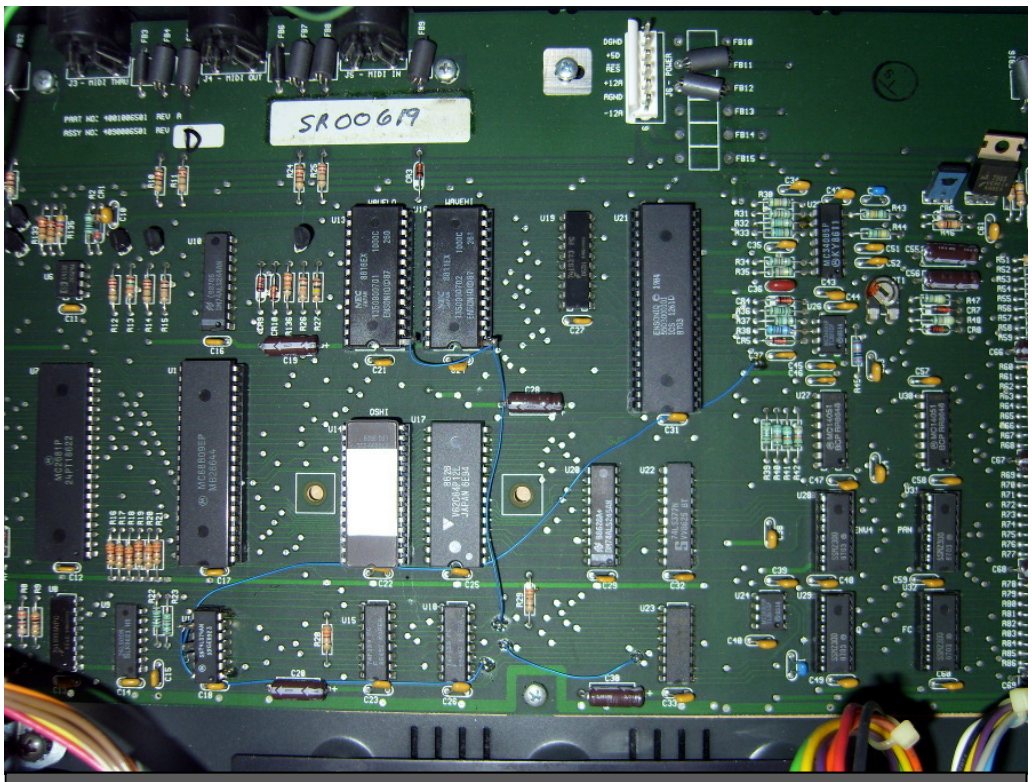
Acknowledgments

Thanks to the efforts of a fellow Ensoniq enthusiast (who I only know by the name Martin), one of those SQ-80m prototypes was tracked down and analyzed. Thanks, Martin, for your enthusiasm and even donating an EPROM reader to the SQ-80m's owner, Kyle Weiss.

Thanks to Kyle for taking the risk of reading out the OS ROM and his efforts of taking pictures of the SQ-80m's main board.

Rainer Buchty

Rainer Buchty is a long-term SQ-80 enthusiast and hardware hacker. He's living in an underground cavern together with over a dozen of 1980's finest synthesizers and samplers. He is known for the world-famous Rainer Buchty Ratio, and, apparently, a super-villain of some sort. But whatever Mark W. tells you, no, we are not twins separated at birth.



MINIMUM-EFFORT SQ80-M | This is Kyle's SQ80-m prototype. Unlike this tutorial, it uses original SQ-80 WaveROMs which due to their custom pinout fit the 27C256 footprint.

Your Mirage, ESQ1, or SQ80 is dead?
Need repair parts?
Drop me a mail: rainer@buchty.net