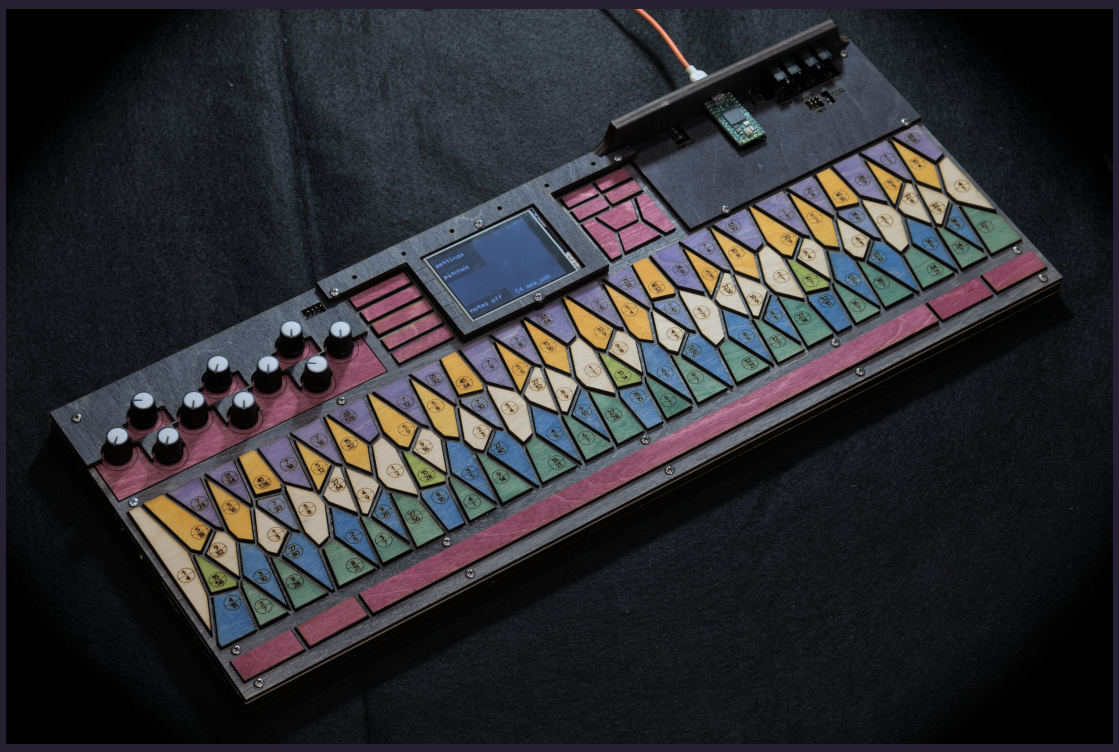
Seven Limit Mosaichord User Manual



Description

The Seven Limit Mosaichord is an expressive musical keyboard controller with four octaves of pressure-sensitive keys, designed around a tuning system called just intonation, in which the frequencies of musical notes are related to each other by whole-number ratios.

The 28-note-per-octave scale includes 16 “5-limit” notes that serve a recognizably similar role as the familiar 12 tone per octave "equal temperament" (12-TET) system used in conventional modern music, plus it has 12 “7-limit” notes for musical intervals, most of which

have no close 12-TET equivalent. In total, it has 113 keys.

The Mosaichord works best with an MPE-capable synthesizer, but we also support a variety of older pre-MPE multi-timbral synthesizers as well by playing each concurrent note on a separate MIDI channel.

MIDI 2.0 is not currently supported.

The Mosaichord is powered over USB, and the default configuration is to send MPE MIDI over USB, to be used by an appropriate software synth. (Surge XT is a good zero cost open source option.) It also has the conventional DIN5 MIDI in and MIDI out ports, to be used with synths that use those.

The keyboard currently requires an external synthesizer to do the actual sound generation, but it *does* have an audio DAC and line-out connection, which may allow us to add that capability in the future.

Music, Math, Just Intonation, and Navigating the Keyboard

As the keyboard layout is unique to this instrument and is based on just intonation (JI), a tuning system that is usually treated as a theoretical curiosity or historical footnote if it is mentioned at all outside of a small community of tuning system enthusiasts, it is necessary at this point to digress for a moment and talk about the mathematical foundations of musical sound and how they relate to the idiosyncratic shapes, physical arrangement, and colors of the keys.

In just intonation, the frequency of every note is related to all the others by some whole-number ratio. In a sense, just intonation is the most basic, universal tuning system there is, as sounds that form whole number ratios are what we percieve as being "in tune". Why this is so has to do with the harmonic series of each note -- the sounds we normally hear in music or just everyday life are not usually pure sine waves, but rather have "harmonics" at

multiples of the base "fundamental" frequency. Thus a sound with a base frequency of 200 hz will usually also be accompanied by sound energy at 400 hz, 600 hz, 800 hz, and so on.

If you were to play another sound concurrently at 300 hz, its second harmonic at 600 hz lines up with the 200 hz sound's third harmonic also at 600 hz, and that sounds very in-tune to us. If you detune, say, the 300 hz sound to 299 hz, then its second harmonic at 598 hz no longer lines up with the other note's harmonic at 600 hz, and it produces what's called a "difference tone" at 2 hz. This is the warbly phasing-in-and-out sound of not-quite-in-tune notes. A little bit of out-of-tuneness is tolerable and sometimes desirable, but usually we'd rather avoid it if we can.

Sounds that are related to each other by small whole-number ratios generally sound the most consonant together, and the larger, more complex ratios sound increasingly dissonant and risk falling into unintelligibility, especially if they aren't played with extremely accurate tuning.

The simplest, most direct and unambiguous musical intervals are often built up from 2:1 ratios (octaves) and 3:2 ratios, as in the previous example (perfect fifths). Tuning systems based on stacking pure octaves and fifths are called *Pythagorean*.

Pythagorean intervals are represented on the keyboard as the row of un-colored "white" keys down the middle. The number printed on the key tells you how it relates to the tonic ($\frac{1}{1}$)and you can see for yourself that these ratios only involve factors of 2 and 3.

Even Pythagorean intervals can become complex, though, if you stack a lot of 2:1 and 3:2 ratios. The Pythagorean major third, for instance, is 81:64. (This interval appears on our keyboard between $\frac{8}{9}$ and $\frac{9}{8}$.)We can get a simpler and more pleasant-sounding major third of 5:4 by allowing for factors of 5 as well. By allowing for the use of intervals with a larger prime number expands the expressive power of the tuning system.

Often we refer to systems of just intonation by the largest prime number factor they use. Pythagorean could be called 3-limit just intonation. When 5:4 became culturally accepted as a "real" note, that marked the transition to 5-limit just intonation. (5-limit JI is also called "triadic just intonation" because it allows for the basic 4:5:6 major triads and 10:12:15 minor triads that are foundational to modern music. In 12-TET those ratios are only a coarse approximation, but the relationship is still strongly implied.)

On the keyboard (assuming the keys that were colored according to the standard color scheme, which might not always be the case), notes that are related to the "tonic" (the $\frac{1}{1}$note in the center) by powers of 5 are either yellow or blue, depending on if you're multiplying by 5 or dividing, respectively.

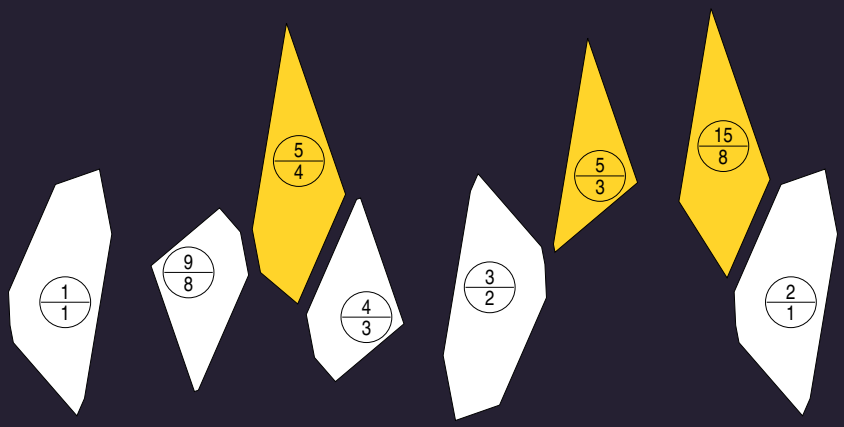
(Note that if you only play the yellow keys or only the blue keys, or only purple and green for that matter, then you'd effectively be in a Pythagorean scale, just transposed from the

white keys by a non-Pythagorean interval.)

To relate the keyboard to a conventional (*Halberstadt*) piano layout, natural (un-stained) and yellow keyboard keys correspond to the white piano keys in the key of C, with the natural keys being the "neutral" scale degrees like the perfect fourth ($\frac{4}{3}$) and fifth ($\frac{3}{2}$), as well as the Pythagorean major second ($\frac{9}{8}$) and minor seventh($\frac{16}{9}$), and the yellows being the the major scale degrees like the major third ($\frac{5}{4}$), sixth ($\frac{5}{3}$), and seventh ($\frac{15}{8}$). The blues are like the black piano keys -- minor third ($\frac{6}{5}$), sixth ($\frac{8}{5}$), and a 5-limit variation of the minor seventh ($\frac{9}{5}$).

You may notice some redundancy -- the piano has twelve notes per octave, but if you add up all the naturals, blues, and yellows, the keyboard has 16 of those keys per octave. That's because it's often necessary in a just intonation tuning system to make distinctions between pitches that are equivalent in 12-tone equal temperament.

For instance, consider the just version of the major scale: C is $\frac{1}{1}$, D is $\frac{9}{8}$, E is $\frac{5}{4}$, F is$\frac{4}{3}$, G is $\frac{3}{2}$, A is $\frac{5}{3}$, B is $\frac{15}{8}$, and we return back to C with $\frac{2}{1}$.



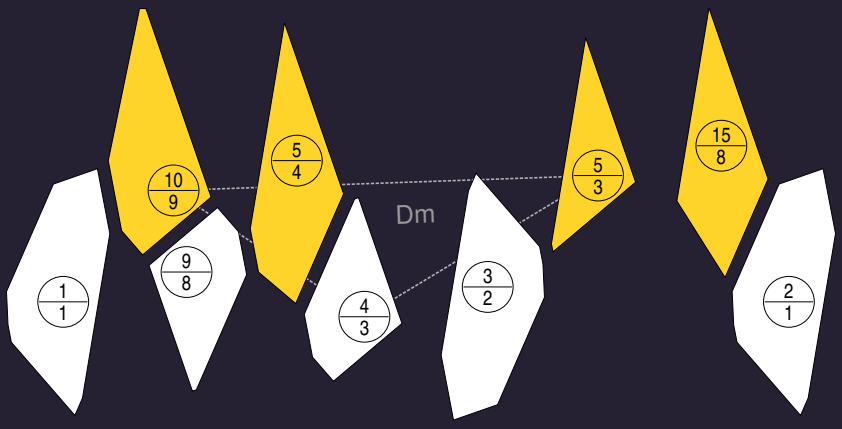
We usually consider the chords in the key of C to be C major, D minor, E minor, F major, G major, A minor, and B dim. A just major chord, as we've said, is made of sounds in a 4:5:6 mutual resonance and minors are 10:12:15. (Octave inversions create some variations on those, but the mathematical relationship is the same if we ignore powers of 2.)

The C major chord is easy, that's just C,E, and G or $\frac{1}{1}$, $\frac{5}{4}$, and $\frac{3}{2}$-- which could just as well be written as $\frac{4}{4}$, $\frac{5}{4}$, and $\frac{6}{4}$. Clearly a 4:5:6 resonance. G major works out nice too. G, B, and D is $\frac{3}{2}$, $\frac{15}{8}$, and $\frac{9}{8}$. We could re-write that as $\frac{12}{8}$, $\frac{15}{8}$, and $\frac{18}{8}$ (shifting the D up an octave to simply things, since inversions are still the same chord), or $\frac{3}{2} * (\frac{4}{4}, \frac{5}{4}, \frac{6}{4})$. The math is a little more complicated, but it works. F is $\frac{4}{3}$,$\frac{5}{3}$,$\frac{2}{1}$ which also works out nicely.

With D minor we encounter a problem. D, F, A is $\frac{9}{8}, \frac{4}{3}, \frac{5}{3}$. This does not make the 10:12:15 ratio of a minor chord. Rather, it makes 27:32:40. If we play it anyways, it sounds off. The

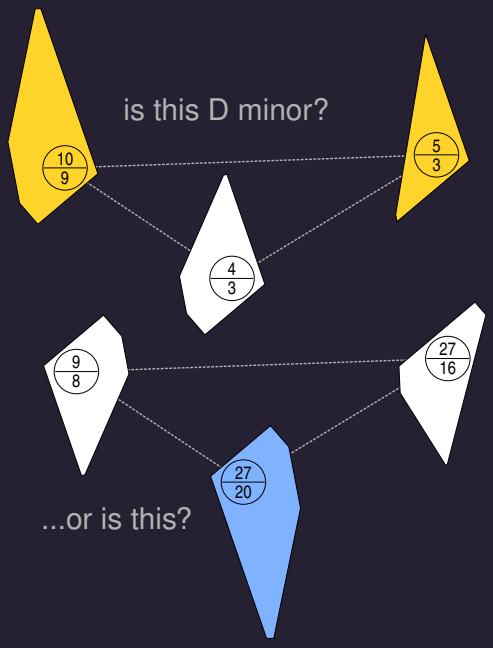
$\frac{4}{3}$and $\frac{5}{3}$make a proper $\frac{5}{4}$major third between them, but $\frac{9}{8}$and$\frac{4}{3}$ make an interval of 32:27, not the 6:5 just major third interval we want.

We can fix the problem by substituting $\frac{10}{9}$ for $\frac{9}{8}$. It's only a little bit flatter than $\frac{9}{8}$. Why not just use $\frac{10}{9}$ all the time as our "official" D note? Well, then the G major chord wouldn't work out because we used $\frac{9}{8}$ there. We haven't left the key of C major, but we find that our simple major scale needs at least 8 notes instead of 7 because there are two slightly different Ds.



We could also have stuck with$\frac{9}{8}$as the root of the d minor chord, but instead introduced $\frac{27}{20}$and $\frac{27}{16}$as new notes to make the 4:5:6 resonance work. That may work too in some contexts, and the 7-Limit Mosaichord has physical keys for all these notes, so you can try them both and make up your own mind.

One challenge of adapting music from 12-tone equal temperament to just intonation is that you have to be aware of these distinctions that didn't exist in 12 tone equal temperament -- which can lead to strange situations like chord progressions that are "supposed" to lead back to the starting chord but actually end up slightly pitch-shifted.



This is related to another issue, that in just intonation you can run out of notes if you go too far in any one direction. There are an infinite number of ratios and we can't put them all on a keyboard with a finite number of keys, so that becomes a limitation of each instrument. 12-tone equal temperament is like a round Earth, where you can travel any direction you want, but keep coming back to where

you started. Just intonation is more like a flat Earth. It's potentially much bigger, and has huge swathes of unexplored territory and fascinating features, but eventually if you go too far you can fall off the edge.

The Seven Limit Mosaichord keyboard (named for its prime limit of seven) is meant to strike a balance between having enough notes to play most conventional 5-limit music as long as it doesn't stray too far from the tonic (e.g. by making complicated key changes), while also making accessible some of the largely unused-in-modern-music intervals that include factors of seven.

As for those seven-limit intervals, the purple keys have a frequency that's related to the tonic by a multiple of seven, whereas the green keys have a frequency that's divided by seven. This can be thought of as the prime seven equivalent of the familiar major and minor of five-limit music, in which the majors are multiples of five and the minors are divided by five.

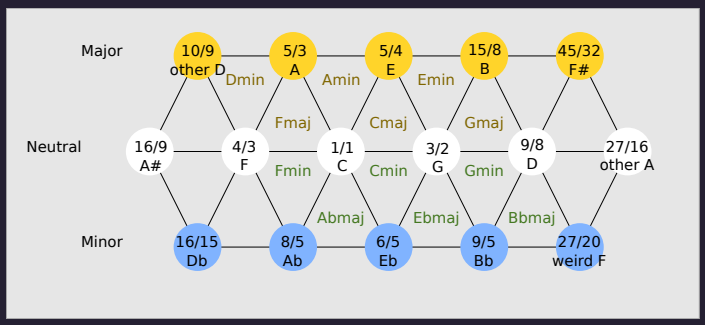
The seven-limit intervals have a characteristic sound that some may recognize from barbershop harmony but is otherwise seldom heard in mainstream music.

The keyboard also has a pair of keys per octave, $\frac{7}{5}$ and $\frac{15}{14}$, which include both a factor of seven and five in the ratio. These are a greenish-yellow and bluish-purple, respectively.

The physical placement of keys is not haphazard, but rather is designed so that musical intervals and chord shapes are consistent anywhere on the keyboard. Transposing a song can be done just by beginning from a different starting point. (That is to say that the keyboard is "isomorphic".) There is a caveat, though, that it's necessary to pick a starting location where you won't run into a situation where some notes that the song requires aren't available on the keyboard because you went too far in one direction and reached the "edge" -- a situation that doesn't come up on equal-tempered instruments.

One useful way to map out the musical territory of a tuning system is using a kind of grid called a lattice or tonnetz. Octaves are usually ignored in lattice diagrams (by convention, fractions are normalized to lie between $\frac{1}{1}$, and $\frac{2}{1}$ by multiplying or dividing by 2) and scale degrees are laid out in a triangular grid where the horizontal axis usually corresponds to perfect fifths, major thirds are usually along the right-leaning diagonal, and minor thirds are usually on the left-leaning diagonal. Generally, to move up and right involves multiplying by 5, and moving right involves multiplying by three. Moving in the opposite directions is the same but dividing instead of multiplying.

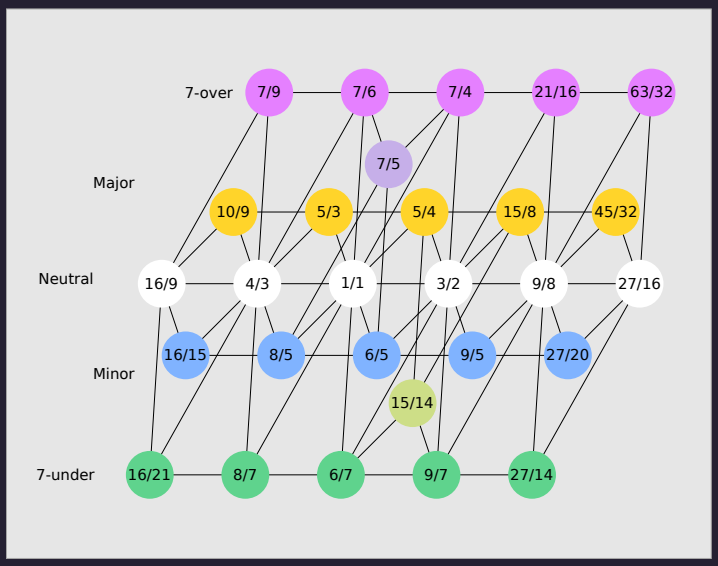
Figure 1:



In this layout, major triads form upward-pointing triangles, and minor triads form downward-pointing triangles. We can also see that there is more than one D, A, and F, and that A♯ and B♭ are shown as different notes.

Displaying the 7-limit notes is a bit more complicated, as each new prime adds a new axis to our lattice, and our diagram has to flatten everything down to 2 dimensions.

Figure 2:



Here we have the 7-limit intervals aadded in, including $\frac{7}{5}$and $\frac{15}{14}$. This can be thought of as a three-dimensional grid with the 7-limit intervals coming out of the page/screen at us. The new long, diagonal black lines represent 7:4 and 7:6 intervals.

The layout of the keyboard is actually very similar to this lattice, save that the lattice ignores octaves whereas the keyboard simply has four partially-overlapping copies of these 28 notes, one for each octave.

(For the curious: the locations of the key centers are determined algorithmically – the horizontal position corresponds directly to pitch, while the vertical position is determined by using a carefully-chosen vertical offset for each prime factor in the ratio. The key shapes are generated by running a Voronoi diagram solver over the previously-calculated key centers. The Voronoi diagram is stretched a bit vertically for better ergonomics and visual appearance.)

That is the general shape of the musical landscape available with this layout; how you use it is up to you.

Basic Operation

The keyboard receives +5v power through a micro-USB cable plugged into the microcontroller board near the upper right. It does not have an off switch; to turn it off simply disconnect power.

It is designed to function as a controller instrument for an external synthesisizer.

To connect to a laptop as a USB MIDI device, the USB connection is all you need. The keyboard behaves as a class-compliant MIDI device -- though not all MIDI synthesizers have the necessary micro-tuning and expression capabilities to use the keyboard as it was meant to be played. An MPE-capable synth is recommended, but some plain MIDI 1.0 devices may work fine too, usually with a little extra configuration. (See the section on device compatibility for more information.)

1. Control knobs – 10 knobs to control synth parameters. Which parameters will depend on the synth in question.

2. Analog input header – used to connect up to four switches or variable resistance devices like expression pedals. (Not currently used.)

3. Status LEDs – 6 programmable RGB LEDs, to be used for status notification, visual cues, or debugging. Current default configuration is for the top left LED glows green when the device is on, and the right-most LED glows when a musical note is playing (more notes is brighter).

4. Menu selection buttons – select one of the menu options displayed on the screen next to it.

5. Screen – displays menu options, navigation, and status. Not a touch screen.

6. Menu navigation buttons – back, forward, ok, and cancel buttons, and dirctional scroll.

7. Stereo audio line out – sends 44.1 khz stereo audio. We may eventually implement an on-board synthesizer, but for now we simply output a sine wave at the current frequency of the tonic note (the one in the center of the keybed labelled $\frac{1}{1}$). The DAC chip we use isn't intended to drive headphones, but it seems to work when I tried it. A preamp may be necessary for some headphones. Currently we output a sine wave at the tonic frequency as a tuning aid. We may add an on-board synthesizer eventually.

8. Microcontroller / usb connection – central processing device of the keyboard, a Teensy 4.0 microcontroller. It runs the device firmware and supplies power from the USB port to the rest of the keyboard.

9. MIDI in/out – standard DIN5 MIDI connectors. MIDI-out sends note, pitch, and expression data to an external synthesizer. Note that DIN-5 MIDI output is normally disabled, as it is slower than USB. Device presets for hardware MIDI synthesizers usually enable it, or it can be enabled manually. MIDI-in is not currently used, though incoming MIDI messages will cause messages to be printed to the the serial terminal.

10. CAN bus – a faster alternative to DIN-5 MIDI. Currently not used, but we've verified that the hardware is able to send and recieve messages. CAN uses two data wires, a "high" and "low" wire. A large number of devices can share the same bus. The keyboard has three high/low pairs, though typically you would only use one or two. CAN bus is usually used in cars rather than to connect musical instruments, but maybe we can start a new trend.

11. CAN termination resistor toggle switch – CAN bus requires a 120-ohm resistor at each end for termination. If the keyboard is connected to one other device or it's the last

device in a chain, then turn this on. Otherwise turn it off. (If, like most people, you aren't using the CAN bus at all then it doesn't matter and you can ignore the switch.)

12. Keyboard expansion bus connector – 9 pin 0.5mm pitch FFC ribbon cable connector for extending the keybed. The electrical design of the keybed makes it at least theoretically possible to daisy-chain multiple keybeds. We haven't tried it yet or even added the necessary support logic to the Mosaichord firmware, but it's an option for future products or ambitious DIY music controller projects.

13. Keys – 113 pressure-sensitive keyboard keys, spanning 4 octaves. The fraction on each key corresponds to the frequency of that note relative to the tonic. For example, if $\frac{1}{1}$is 100 hz, then $\frac{3}{2}$would be one and a half times that, or 150 hz. Colors correpsond to the prime factors 5 and 7 that occur in the top or bottom of the fractions.

14. Octave transpose buttons – transpose base pitch up and down by octaves.

15. Pitch bend bar – an extra long control surface for pitch bend. Unlike the other keys and buttons which act as single pressure sensors, this one has two pressure sensors, an upper and a lower one. Horizontal position doesn't matter, but it can sense up-and-down variations in pressure. It is designed very much like the space-bar on a typewriter-style computer keyboard to be within easy reach of one or both thumbs at any time.

16. Semitone transpose buttons – transpose base pitch up and down by standard 100-cent 12-tone equal tempered semitones. As with the octave transpose buttons, the pitch of the tonic $\frac{1}{1}$ key in the center of the keyboard is displayed on the screen. (We may add some additional options later, such as transposing by 41-EDO semitones.)

Connecting to external synthesizers

For most electronic musical instruments made in the last approximately 40 years, interoperability between keyboards and synthesizers is pretty straightforward – just plug a DIN5 cable in to the MIDI-out port on the controller and the MIDI-in port of the synthesizer, set them both up to the same channel, and it just works. Or if it's a software synthesizer, then connecting a controller or synth to a laptop with a USB cable is much the same.

We'd like to say the process of setting up the Mosaichord with any MIDI synthesizer was that simple, but as the keyboard makes use of continuous key pressure and requires some very specific tuning capabilities, there are a great many otherwise amazing synthesizers

that simply aren't compatible, and the ones that are may require some special attention to configuration details.

We've provided presets for a handful of synthesizers we've tested with the Mosaichord. It would be a monumental task to try to have presets for every MIDI synth ever made, but we hope that the presets we do have will be useful to the people who already own one or two of these synths, and provide a good starting point to add more as time allows.

The pre-defined "device presets" for various synthesizers can be found in the Mosaichord menus under SETTINGS→OUTPUT→DEV PRESETS.

MPE Synths

Fortunately, the capabilities we want have been standardized in recent years as an extension to the MIDI protocol called MPE. Using the keyboard with MPE-capable synths is the easiest option.

MPE controllers are expected to send a handshake when they connect, to tell the synthesizers that it's using MPE and how many channels to use. The keyboard sends this when booting up or when switching to an MPE-enabled preset, but it's also possible to send it explicitly by invoking SETTINGS → OUTPUT → MPE INIT.

Surge XT

Surge XT is probably the easiest option to get up and running quickly. It is a free and open source MPE-capable software synthesizer that runs under Windows, MacOS, and Linux.

A few things are worth paying attention to. Enable MPE by clicking on the MPE button. The MPE pitch bend range should be set to 48 semitones. (This is in the menus and is separate from the "regular" bend depth setting near the upper left of the main window.)

Better tuning precision can be achieved with a narrower MPE pitch bend range so feel free to change it, just be aware that the keyboard and Surge have to agree on what the range is or the sounds that come out will be out of tune.

Most of the regular patches aren't designed to react to MPE key pressure, but there are a handful of MPE-specific patches including some contributed by Roger Linn for the Linnstrument. (The Mosaichord keys lack have a per-note "Y axis" control that the Linnstrument uses to affect MPE Timbre, though it's possible to adjust the timbre for all the notes at once using knob 8.)

The non-MPE patches can often be adapted for more expressive control by setting MPE PRESSURE as a modulation input for Amp Gain, the cutoff for filter 1 or filter 2, or any number of other controls.

When connecting the keyboard over USB, depending on operating system you will likely have to select SEVEN LIMIT MOSAICHORD under audio/midi settings in the OPTIONS menu in the top left.

We have provided a Surge XT device preset in the menu under SETTINGS→OUTPUT→DEV PRESETS→SURGE XT, but it's also the default preset on boot up.

(This preset assumes the connection will be over USB. You can use DIN5 instead by manually enabling SETTINGS→OUTPut→DIN5 MIDI.)

This device preset may work with other MPE-capable synths, but we haven't tested.

Multitimbral MIDI Synths

Many synthesizers that don't support MPE can still be used with the Mosaichord, but there are a number of things that can go wrong if the synthesizer and the Mosaichord are not configured correctly, and even when they are they may require some trial and error to find what works best.

The most common thing to go wrong is that the tuning is off. To understand how this happens, it helps to know how the Mosaichord is able to use arbitrary tunings on synths that, for the most part, were never designed to support it in the first place.

Pre-MPE MIDI 1.0, originally standardized in 1983 and used ever since, doesn't have any explicit support for tuning. (Well, technically it does. There's MTS, the MIDI Tuning Standard ratified in 1992, but it's not widely supported.)

What MIDI 1.0 *does* have that is supported by almost all MIDI synths is pitch bend. It even has a native resolution of 14 bits, which is enough for pretty accurate tuning even if the pitch bend range is very wide. The main problem with pitch bend, though, is that it's not applied on a per-note basis. Rather, it affects *every* note that's currently being played in the current channel. That's not very useful if we want to play a chord and have every note perfectly in-tune (according to a definition of "in tune" that differs substantially from 12-tone equal tempament.) We want to be able to bend every note individually.

It's possible to have per-note pitch bend, though, if we constrain ourselves by only playing one note at a time per MIDI channel. MIDI has 16 channels, so that gives us 16 voices of polyphony, which is pretty reasonable. We can also use other channel-affecting MIDI CC ("Continuous Controller") messages on a per-note basis, like volume or filter cutoff.

Not all synths support multiple channels. The ones that do are called “multi-timbral” as the feature allows the synth to produce several different timbres at the same time, as if it were multiple separate instruments. Most analog polyphonic synths are unfortunately mono-timbral, but digital synths that use ROM-based sample presets (often called “romplers”) were often multi-timbral.

This way of doing things has some drawbacks, though: the Mosaichord controller must send MIDI commands to the synth without knowing if they’re being interpreted correctly, and that interpretation depends on settings like the pitch bend range. If it thinks it’s something different than what it is, the notes will be either jarringly out of tune if it bends too far, or somewhere between correct tuning and 12-EDO if it doesn’t bend far enough. We might even have have situations where a synth is configured with a different pitch bend range on different channels.

The pitch bend range defaults vary by synth, or even between preset patches on the same synth. Two semitones up and down is a common default. Fortunately there is a standard MIDI RPN to set a channel’s pitch bend range explicitly, and most synths we’ve tried support it.

Besides having to be careful about pitch bend range configuration, using multiple channels creates other inconveniences. If we want to change a patch or some sound engine setting, we have to make the exact same setting change to all 16 channels. We *could* do this manually, but it would be tedious, so instead we integrate much of that functionality into the Mosaichord, so you can make the change once and it propagates that change across all the channels.

The most common use of this is changing the active patch preset. Most synths have banks of presets arranged into convenient categories like “keyboards”, “strings”, “brass” and so on. At the level of MIDI messages, though, we deal in terms of “banks” which can have up to 128 patches selected with “program change” messages, and we can select what bank is active by sending a 14-bit number split into 7 “most significant” MSB bits and 7 “least significant” LSB bits. That gives 16384 possible banks. Most rompler synths of the 1990’s and 2000’s have maybe somewhere around half a dozen or so, and the rest are empty space. Where the real banks are in that large address space varies greatly by manufacturer and model, but it’s usually listed somewhere in the user manual.

The device presets we’ve defined usually impose an upper and lower bound on the MSB and LSB to make it easier to find the populated banks.

MSB, LSB, and patch preset (i.e. Program Change) numbers can be found in the Mosaichord menu under PATCHES. Select the value you want to change, and press the right and left direction pads to increase or decrease the value. Changes are sent immediately. You can

also “unlock” the MSB and LSB range limits imposed by the device preset.

Note: we count patches from 0-127, whereas some manufacturers count them from 1-128.

E-mu Proteus 2000

The Proteus 2000 (released in 1999) is a great synth to use for microtonal music, as it allows user-defined tuning tables. We don’t use that feature though, because the one-note-per-channel pitch bend trick accomplishes the same thing. (Maybe some day.)

Using pressure to modulate volume seems to work pretty well on the Proteus 2000 with most patch presets, so we leave it enabled by default in our “Proteus 2k” device preset. In particular, this synth applies reverb *after* channel volume rather than before, which is nice as it means the reverb tails aren’t unnaturally affected by key pressure.

Korg Trinity

The Trinity (released 1995) was available with a keyboard, or as a rack unit called the TR Rack. Connect the MIDI-out on the Mosaichord to the MIDI-In on the Trinity.

To use all 16 MIDI channels, press the GLOBAL/MULTI button a couple times to enable “Multi” mode and select the Korg Trinity device preset on the Mosaichord under SETTINGS→OUTPUT→DEV PRESETS→KORG→TRINITY

Using pressure for key volume doesn’t seem to produce a good result on most preset patches, so we leave it off by default.

Roland XV-2020

The XV-2020 is a multi-timbral (mostly) sample-based MIDI synth released in 2002, as a continuation of their JV/XV product line.

To connect to the Mosaichord, connect a MIDI cable from the MIDI-out of the keyboard to the MIDI-in of the XV-2020, turn the XV-2020 on, and press the “value” button until there is a single red light by the “GM” label. That puts it into “General MIDI” mode, but also sets it to receive on all channels. General MIDI is a bank of standard instruments, so that, for instance, patch 0 is always a piano, patch 40 is always a violin, and so forth. (Here, we’re counting from 0-127, but often devices and documentation will count from 1-128. We don’t have to use the General MIDI sounds though, we can select a different bank by changing the bank MSB (“most significant bits”) and LSB (“least significant bits”).

Onece the XV-2020 is up and running in GM mode, select the XV-2020 preset on the Mosaichord via SETTINGS→OUTPUT→DEV PRESETS→ROLAND→XV-2020.

By default, we set the MSB to 87 and allow LSB to vary from 64-67. This accesses the internal preset banks. To access the user presets and any expansion cards will require unlocking the MSB/LSB range. Select PATCHES→UNLOCK in the menu.

We have found that modulating channel volume according to key pressure on the XV-2020 doesn't usually create a satisfactory result (using both key pressure and velocity for volume at the same time generally don't go well together, and the XV-2020 has a tendency to lag when it receives too many MIDI CCs in quick succession), so we turn it off by default on this preset.

The XV-2020 supports polyphonic aftertouch. This is disabled by default on the Mosaichord, but you can enable under SETTINGS→CONTROLS→POLY AT.

Yamaha FB-01

The FB-01 works surprisingly well with the Mosaichord, considering it was released in 1986. Connect the MIDI-out of the Mosaichord to MIDI-in on the FB-01, and select the device preset under SETTINGS→OUTPUT→DEV PRESETS→YAMAHA→FB-01.

The device preset is configured to do some unusual things that are specific to the FB-01. The FB-01 has 8 voice circuits, and these are statically assigned to MIDI channels. Usually, a user would assign them all to a single channel, but in our case we want to assign a single voice to each of the first 8 channels. We do this by sending some MIDI SysEx commands to set each MIDI channel and "note count". If this doesn't work, you may have to manually configure the FB-01 from the front panel. (One reason why this might not work is that the FB-01 system channel is set to something other than 1. "System channel" is something specific to the FB-01 and is independent of MIDI channels.)

The FB-01 does not use the same MIDI CC MSB/LSB system for loading banks of preset patches that became standard on later MIDI hardware; instead it uses a special bank select SysEx message. We've set the FB-01 device preset so that it sends that other SysEx message, and changing the bank LSB as normal on the Mosaichord sends that other SysEx message instead.

Also, when changing patches we set the pitch bend range explicitly each time (as the pitch bend range otherwise varies by patch), and we explicitly set "detune" to 0 and disable the LFO, as it usually detracts from the sound more than it helps when working in just intonation. (Most patches have some vibrato that you can add back in using the far left knob 4 on the Mosaichord, which acts as a mod wheel.)

Yamaha MOX8

Released 2011, the MOX8 works with the Mosaichord with a bit of setup. First, set it to receive on all channels by pressing the "SONG" button. Connect the MIDI-out on the Mosaichord to MIDI in on the MOX8, and select the MOX device preset under SETTINGS→OUTPUT→DEV PRESETS→YAMAHA→MOX8.

On the MOX8, you can press the "MIXING" button then F2 to see what patch is active on each channel. These should all be the same.

Depending on what preset was previously active, you may find, though, that the channels don't all *sound* the same. They might all have different values for filter cutoff, sustain, reverb send level, or any of the other myriad sound settings.

These settings can be checked manually and set by pressing each "PART" button over on the right of the MOX8 control panel and then selecting/adjusting each control over on the left side of the MOX8 control panel. For convenience, we use MIDI CCs to set *most* of these to reasonable defaults when the MOX8 device preset is selected.

MIDI → CV devices

Sometimes we might want to use the keyboard with synthesizers that work based off of analog control voltage signals and gate inputs rather than MIDI. For that, there are MIDI to CV devices that will do the necessary conversion for us.

Arturia Keystep Pro

The Keystep Pro (or KSP) in addition to its many other uses can serve as a MIDI to CV converter, to interface with a modular synthesizer. The KSP does not, as of the time of writing this, support MPE, but it does allow us to use a separate channel for each voice, which is good enough for our purposes.

Used as a MIDI to CV converter, KSP supports up to four channels of polyphony. Each channel has its own dedicated pitch CV, modulation CV, and gate output. The pitch CV is affected by MIDI pitch bend.

A conventional setup would be to run the four pitch CV outputs into four identical voltage-controlled oscillators (VCOs), the modulation CV inputs would be connected to four identical voltage-controlled amplifiers (VCAs) controlling the volume of the four VCOs, and the gates would trigger envelope generators to further modulate volume and/or filter cutoff.

Getting a good result will depend a great deal on how accurately the VCOs track pitch CV and how well they stay in tune. I've had decent luck with 3340-based VCO designs, but they require careful calibration for best results.

The KSP settings can be configured from a computer using Arturia's MIDI Control Center. (Some of the settings may be configured on the device itself, but there are some important ones that can't as of the time this is written.)

You'll want to configure the KSP's four voices to be on MIDI channels 1, 2, 3, and 4 with a pitch bend range of +/- 12 semitones. The CV output for all channels should be set to 1v/octave (unless you're using a modular system based on a different standard) and the Modulation CV out for all four channels should be set to "mod wheel" and should all have the same output voltage range.

From there, connect a MIDI cable from the MIDI-out port of the Mosaichord keyboard to the MIDI-in port of the KSP, select the SETTINGS→OUTPUT→DEV PRESETS→ARTURIA→KEYSTEPPRO device preset using the Mosaichord menu, tune your oscillators as needed, and you should be good to go.

(The sine wave output from the line-out port on the Mosaichord can helpful as a reference pitch to tune against.)

Untested

- Mutable Instruments Yarns
- Expert Sleepers FH-2

Unsupported

- Korg SQ-64 (no support for pass-through of pitch bend when used as MIDI to CV converter)

Mono Synths

Monophonic synths are, in most respects, easier to support than polyphonic synths, as we only need to manage a single channel. We can generally just use the front panel controls normally.

Monosynths tend to have some sort of note-priority logic – usually either the highest note, or the lowest note, or the most recent note is the one that's actually played.

Since we need to know which note is currently being played in order to send the right pitch bend command, we make sure to only have one active note at a time, by sending a note-off before the next note-on. We use most-recent-note priority.

Moog Slim Phatty

The Slim Phatty (released in 2011) is pretty straightforward to use. Unlike most analog mono synths, it provides MIDI control over most of the synth parameters.

One thing to be careful of is that preset patches can override the pitch bend range. We send a MIDI RPN to reset pitch bend range when selecting a preset patch from the Mosaichord menu system, but if you change presets from the front panel of the Moog, the pitch bend range could get off from what the Mosaichord thinks it is. You can fix this by either selecting presets from the Mosaichord, or by resetting the pitch bend range (SETTINGS→OUTPUT→BEND RANGE).

The Slim Phatty responds to MIDI CC 11 (channel volume) but it produced some audible artifacts when I tried it, so we default to using CC 19 for pressure, which affects filter cutoff.

Tuning Table Synths

Some mono-timbral synths can be used with the Mosaichord by loading a custom tuning table. The details of how exactly this is done vary a great deal between different brands and models of synthesizer.

There *is* a tuning table standard that was eventually added to the MIDI standard called MTS, but we don't currently support it.

(Some multi-timbral synths that support custom tuning tables, such as the Yamaha FB-01 and the Emu Proteus 2000, already work with the Mosaichord by using multiple channels and pitch bend.)

Yamaha DX7 with Grey Matter Response E! expansion board

This one is a bit unusual. The original DX7 did not have custom tuning table support (though it was available in the later DX7IID and DX7IIFD). However, Grey Matter Response developed a popular add-on board for the DX7 that essentially replaces the original firmware

with a custom firmware and adds additional patch storage memory. Among the new features supported by the E! firmware is custom tuning table support.

To use the Mosaichord with a DX7 with the E! expansion, connect the MIDI-out of the Mosaichord to MIDI-in of the DX7, and turn them both on. By default, the E! firmware blocks incoming SysEx messages, so you'll have to turn off the midi SysEx filter. (Hit "function" then "operator select" several times until it says "keyboard control" then hit the "2" button and use the value slider to turn Sys/Ex on.) Then load the E! device preset by selecting SETTINGS→OUTPUT→DEV PRESETS→YAMAHA→DX7 WITH E! in the Mosaichord menus.

If the tuning table is loaded correctly, the keyboard will sound the notes with their proper just intervals. If for some reason the tuning table doesn't load, the pitch range will be very widely stretched, as the 113 keys will span over 9 octaves, with each key's pitch separated by an equal tempered semitone.

The DX7 will be able to respond to note-on velocity, but not key pressure. (We would use MIDI 1.0 polyphonic key pressure messages for this, but unfortunately the E! firmware does not respond to those.) For more expressive playing, we'd recommend the Yamaha FB-01 – despite its more primitive sound engine (4-operator fm versus 6-operator in the DX7), we can modulate the volume of individual notes by spreading them across 8 MIDI channels and using channel volume CC messages.

Semitone and octave transpose features work as usual with the DX7 with E! device preset, though be aware that the Mosaichord sends a 521 byte SysEx message to update the tuning table whenever the transpose setting changes, so it may introduce some slight latency.

The E! tuning table format is different from the one used by the DX7IID and FD, so this device preset will not work with those synths.

Dexed

Dexed is an open source re-implimentation of the DX7 sound engine, and can be used to play DX7 patches on a computer. Dexed supports MPE, but we were unable to get it to recognize pitch bend commands. However, it does support custom tuning tables via Scala scale files (**.scl*) and keyboard maps (**.kbm*). If we load the appropriate tuning table, Dexed can work in pretty much the same way we use the DX7 with the E! expansion.

To use Dexed with the Mosaichord, connect it to your computer with a usb cable, run Dexed, and select the Mosaichord midi input device (however that is done on the operating system you're using).

On the Mosaichord, select the SETTINGS→OUTPUT→DEV PRESETS→DEXED device preset from the menu. Within Dexed, click the PARM button and input the *seven-limit-mosaichord.scl*

and *seven-limit-mosaichord.kbm* files available from the "*extras*" subdirectory in our source code repo here:

`https://github.com/jimsnow/microtonal-controller/tree/main/extras`

Enable the transpose "12 as SCL" option, otherwise some patches are likely to sound quite strange. Leave the MPE setting off[1].

The transpose buttons on the mosaichord will not work with the DEXED device preset, but Dexed has a transpose option in the interface. You can also edit the *.kbl* file as desired.

[1] You can always try out MPE if you really want to to see if you get better luck than we did, but in that case don't load the .scl or .kbl file and stick with the SURGE XT device preset instead of the DEXED device preset.

Firmware updates and modification

While we hope that the Mosaichord works just fine for most people in its stock form, this is a new product and we expect to be continually adding software features over time, and we have released the firmware as an open source project to enable our users to make changes to suit their preferences as well, if they so desire, as this is ultimately *your* instrument not ours.

It is also not possible for us to develop and test device presets for every kind of synthesizer, and we expect there will always be some obscure configuration options that aren't available in the menus and can only be accessed by modifying the program directly.

For all these reasons it may, from time to time, be necessary to update the the device firmware.

Install/Setup development tools

The Mosaichord uses the Teensy 4.0 microcontroller, and firmware development is normally done using the Arduino IDE.

Arduino can be downloaded from `https://www.arduino.cc/en/software`. (Under linux, you may be asked to tweak some serial group permissions.)

Support for the Teensy microcontroller can be added from within the board manager of the Arduino IDE, once you've added a URL to the board manager settings.

Under FILE→PREFERENCES→SETTINGS, add `https://www.pjrc.com/teensy/package_teensy_index.json`to "additional board manager URLs".

Search for Teensy and install that package using the board manager (left pane of the main IDE window).

Set TOOLS→USB TYPE to SERIAL+MIDI.

(More detailed instructions on installing the Teensy toolchain are available here:`https://www.pjrc.com/teensy/td_download.html`.)

Getting Source Code

Download the latest source code from `https://github.com/jimsnow/microtonal-controller` (I'd recommend the "git clone" method, as it makes downloading new updates easier, but downloading a zip file should work as well if you don't have git installed on your system.)

Open the file *microtonal-controller/src/microtonal-controller/microtonal-controller.ino* in the Arduino IDE.

At this point you can make any changes to the source code if you so desire.

Either way, you can connect the Mosaichord to your computer with a USB cable and click the right arrow button near the top left of the Arduino IDE window to compile and upload the firmware to the Teensy.

You may in some circumstances need to push the small white button on the teensy to put the bootloader into a mode where it waits for new firmware to load.

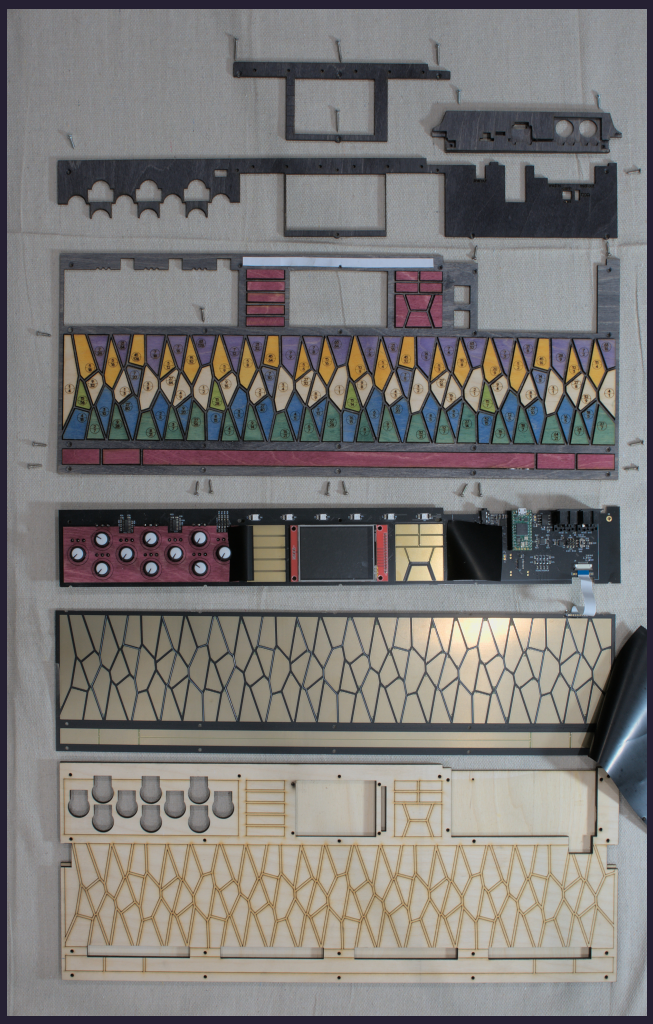
If all goes well, some red lights will flash on the Teensy and then it will reboot into the latest Mosaichord firmware.

Debugging

One of the very useful features of the Arduino IDE is a serial console. This is used by the Mosaichord firmware to print diagnostic messages. It can be enabled in the IDE under "Tools→SERIAL MONITOR".

Usually the output isn't very verbose, but more output can be enabled with various options in the Mosaichord menus under SETTINGS→DEBUG. "SHOW RES" for instance displays current values being read by the 160 inputs to the ADC matrix (most of them correspond to keys, but some are for the control buttons, knobs, fixed calibration resistors, and the analog input headers), as well as resistance values before and after applying a calibration formula.

Assembly and Disassembly



Sometimes it may be necessary to take the Mosaichord apart to fix something. Most often this will be to either clean oxidation off the printed circuit board (PCB) traces, causing some keys to be less responsive to pressure than they should, or to remove some foreign debris that managed to get caught between the PCB and force sensitive resistor film shorting a connection and causing a stuck note or button. These are both straightforward to remedy, but require disassembling the keyboard.

A small phillips screwdriver is all that's needed for assembly and disassembly, though some clear adhesive tape may come in handy if you need to replace any of the existing tape.

Isopropyl alcohol is what we use to clean the PCB when necessary.

To disassemble the keyboard,

- disconnect from USB power
- remove two screws holding on the small back plate and remove it
- remove the 20 oval-head screws holding the bezel pieces, front panel, and key tops in place
- gently pull the front panel off (the screen may be a tight fit)

And that's pretty much it. It isn't necessary to remove the knobs unless there's something wrong with the potentiometers or you need to replace the control PCB for some reason.

There are three force-sensitive resistor (FSR) pieces – one large one that covers the whole lower PCB, and then two smaller pieces on the control board to either side of the screen. This black plastic material[2] is called Velostat, and is typically used to make anti-static bags. It also has the useful property that it has a certain electrical resistance that reduces when pressure is applied, which makes it very useful as a force sensor.

It is also possible to play the keyboard with nothing more than bare fingers directly on the PCB, but that requires very different calibration settings and results tend to vary with humidity.

The FSR should be held in place by a small amount of tape. If you remove the FSR to clean it or the PCB, be careful not to touch the exposed gold traces on the PCB under the FSR, as finger oils will eventually cause oxidation which will need to be cleaned off again some time in the future. We usually scrub the PCBs off with 70% isopropyl and paper towels, then let it dry and get all the lint off before taping the FSR back down and reassembling.

If it's ever necessary to replace the upper or lower PCB, you'll need to disconnect the ribbon cable[3] between the two. This is easily done by pulling out the slide-lock mechanism on either end. The ribbon cable can be replaced by inserting the cable back in and then engaging the slide lock. (The ribbon goes in with the blue side up. The cable should not twist upside-down.)

The Teensy 4.0 microcontroller is in a socket and can be carefully pulled out and replaced if necessary. (A Teensy 4.1 should work just as well, though it will overhang the socket by quite a bit and will require modification to the bezel or just leaving it off entirely. It's possible to modify another socket and solder it down to the pin holes provided to act as

[2] Mouser part number 517-1704-36

[3] 1mm pitch 9 conductor FFC cable with opposite-side contacts, Mouser part number 538-98267-0772

extra support. Remember to select the Teensy 4.1 board instead of 4.0 in the Arduino IDE when compiling/loading new firmware.)

The screen is soldered in place. Replacing the screen can be done with a hot-air rework station, but it's not very easy.

Assembly can be done in the reverse order as disassembly, but it's usually a good idea to power up the Mosaichord and test it out once you have about three or four screws in, just to make sure the keys are all working right. It's much easier to identify problems and fix them before you've put all the screws back in place. I find it often takes two or three tries to get everything right when assembling a new Mosaichord.

Theory of Operation

It isn't necessary to know how the Mosaichord works to use it. However, some understanding may provide useful context for anyone who wants to make significant software modifications, and also some people just like to know how things work. Therefore we will explain the basic principles here. Some familiarity with basic electronics is assumed.

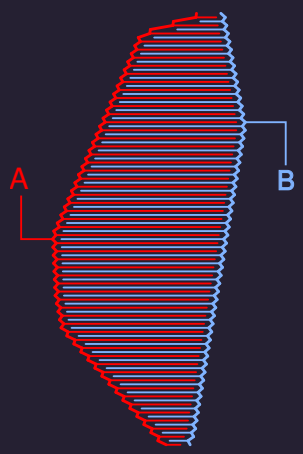
The schematic is not included here as it may change from time to time, but it can be found alongside the firmware source code.

Keys

A good place to start is: how do the keys work?

Each key has a pressure sensor underneath it that normally has a fairly high electrical resistance. Pressing the key causes the resistance to drop – the firmer the pressure, the lower the resistance.

How these pressure sensors work is simple; on the printed circuit board we have two sets of gold-plated copper traces that are like fingers that nearly mesh but do not actually touch each other. Over the top we lay a force-sensitive resistor (FSR) material. The FSR connects both sets of traces (labelled A and B in the figure). The resistance between A and B is high when pressure on the FSR is light, but increasing pressure reduces the resistance, allowing electricity to flow more easily.



If we connect A to 3.3 volts through a pull-up resistor and connect B to ground, A will stay near 3.3 volts when pressure is light, but the low resistance path to ground will cause the voltage to drop much lower under high pressure.

The microcontroller we use has a number of analog input pins we can use to measure voltages, using one of several on-board analog-to-digital converters (ADCs).

Multiplexing inputs

Of course it's not quite as simple as wiring each key to a pin on the microcontroller. The keyboard has 113 keys in addition to a handful of buttons and knobs that we need to read

as well. The microcontroller we use, the Teensy 4.0, only has 14 pins capable of analog input. Clearly we need more inputs than that.

There are a couple standard ways to deal with this, that have their pros and cons. One option is to use a key matrix. Keys are organized into a grid of rows and columns. You get a reading on a given key by measuring the resistance between its row and its column, with all other rows and columns disconnected from power and ground.

The matrix approach conserves on hardware, but it tends to suffer from "ghosting" where if you press three keys that are all corners of a common rectangle, the fourth corner will appear to pressed as well. This problem can be reduced or eliminated by connecting each key through a diode, but that's a lot of diodes (eliminating some of the benefits of using a matrix to begin with) and diodes introduce a voltage drop — generally around 0.7 volts, or 0.3 volts for Schottky diodes. Since we're inferring pressure from the voltage we read, we need that to be as accurate as possible. Even 0.3 volts creates a pretty big "blind spot."

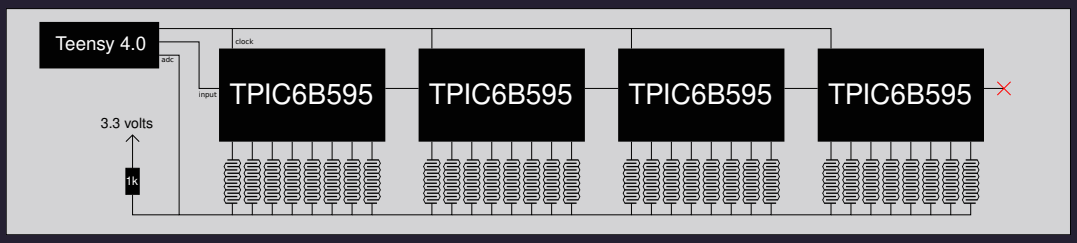
Another approach is to use multiplexer chips. These are basically IO expanders that work like a selector switch that's controlled with some address pins. For example, a 16-way multiplexer would have 4 address pins used to encode a binary number that selects which of 16 input pins are connected to a single output. (Or vice versa. Multiplexers are usually bi-directional, and are often designed to work with analog signals.)

We think this is a good approach. It would probably work fine. It's not what we did, though – in the Mosaichord we use shift registers.

Normally, shift registers are used in digital systems to convert parallel data to serial (in an "input" shift register) or serial data to parallel (in an "output" shift register). The type we use is an output shift register, specifically the Texas Instruments TPIC6C595DR.

Output shift registers generally have an serial input pin and a clock input. (Actually, there are two clock inputs, but we drive them both at the same time from the same microcontroller pin.) There are 8 "regular" outputs, plus one extra.

A shift register can be thought of as an 8 bit memory device. Each output pin is either on or off depending on the stored bit, and pulsing the clock causes all the "bits" to shift over one place to the right, with the input pin determining whether the left-most bit is a 1 or a 0. If we need more than 8 bits, we can just connect the extra output of our first shift register to the input of another, and wire all the clocks together. We can chain as many together as we need in this fashion.



We chose the TPIC6C595 in particular because of an interesting feature that makes it useful in this context. Instead of the traditional style of style of shift register where output pins emit the supply voltage when “on” and 0 volts when “off”, these have open-collector outputs. What that means is that when a pin is “on” it has a low resistance path to ground, and when the pin is “off” it’s in a “high-Z” state that’s effectively not connected to anything.

What this allows us to do is connect one side of all the pressure sensor PCB traces together, pulled to 3.3 volts through a 1k pullup resistor. We connect the other side of the pressure sensor traces each to its own shift register pin.

We manage the shift register so that at any given time, only one pin at a time is “on”. That means the key connected to that pin is the only path to ground in the whole circuit – none of the other keys matter, because none of them are an electrical path to anywhere. The variable resistance of the key pulling the voltage down to zero fights with the pull-up resistor pulling the voltage up to 3.3 volts. We can connect the trace that connects half of all the pressure sensors to an ADC input on the microcontroller and take a voltage reading, and from that we can infer the resistance and therefore the key pressure.

The main loop of the program running on the microcontroller basically just scans all the keys in order over and over.

We’ve glossed over a few complications: we have an op-amp buffer in front of the ADC input to (we hope) make the voltage settle faster, we use a level-shifting IC between the microcontroller and the shift registers because they run at different voltages, and we don’t actually connect all the keys to a single ADC line – we actually have four independent ADC lines, each with its own pull-up resistor and ADC input pin. We call these ADC channels – not to be confused with MIDI channels. (The Teensy 4.0 has two hardware ADCs, meaning we can only do two readings at a time simultaneously. However, having four separate circuits reduces the amount of noise and capacitance that any one circuit would have. Even though only one key at a time is productively doing anything, the rest can still potentially act as antennas and otherwise degrade signal quality.)

The upper PCB that has the microcontroller, I/O, and user interface controls has four shift registers, one for each ADC channel.

The seven-limit four-octave keybed has another 16 ADCs, 4 each per channel.

Effectively, that gives us 20 times 8 or 160 inputs. Not all of these are connected to keys. Some are connected to potentiometers, or fixed resistors, or in a few cases, nothing at all. Each ADC channel is responsible for 40 inputs.

We can do a complete scan of these 160 inputs about 500 to 900 times per second, depending on whether we care more about speed or accuracy.

One of the benefits of using shift registers as opposed to multiplexers is that there's no hard upper limit on the number of inputs we can have. That means we can use the upper "control" PCB with keyboards of practically any design, with more or less keys. We can even chain multiple of the keybed PCBs together (though we haven't actually tested this).

One feature of the keybed PCB is that the first shift register pin on each channel is connected to a fixed resistor. The purpose of this is that we can give each keybed PCB variation a different set of fixed resistors, so the control board can identify it. If the keybed isn't connected, the resisters will all show up as infinite resistance. Thus we can auto-detect when a new keyboard is connect to the chain simply by reading one bit past the end of the last shift register we know about and see if we see a resistor pattern we recognize.

Building a custom keyboard or expressive controller that chains off of the Mosaichord should be fairly feasible as a DIY project, and in fact that's part of the reason why we describe in such detail how keyboard works here.

Autocalibration

One of the difficulties with the design as described above is that the four ADC channels aren't actually as independent from each other as we would like – readings on one channel tend to affect readings on other channels, sometimes significantly. The reason why this happens is that the force-sensitive resistor material creates a sort of electrical path between things that aren't *supposed* to be electrically connected, simply because they're physically near each other and they're both touching the FSR. What's worse is that the resistance of these unwanted electrical paths changes depending on what keys are being pressed and how hard.

We compensate for this by measuring the resistance between each of the four ADC channels on each keyboard scan. We then use the measured resistances to infer what the voltages on each channel should be for that particular set of four inputs if the four channels weren't pulling each other towards some sort of average value.

This seems to work well enough. We may be able reduce the magnitude of the problem in the future by increasing the width of the thin strip of solder mask that separates the keys on

the keybed PCB.

Generating key pressure from MPE

Turning key pressure readings into MIDI commands (not to mention the autocalibration routines mentioned previously) is done by the firmware that runs on the microcontroller.

Basically, the main loop keeps track of whether the pressure of each key exceeds a certain minimum activation pressure – if so, it grabs an unused MIDI channel (or steals one, if none are available), and sends the appropriate pitch bend and note-on commands to begin the note.

On subsequent scans, if the key pressure remains above the minimum threshold, the program does not send additional note-ons, but rather it sends key pressure commands according to the MPE spec (if MPE key pressure is enabled) or it sends polyphonic aftertouch (if poly AT is enabled).

If the pressure drops below the minimum threshold, then a note-off is sent.

There are some additional details, depending on what settings are enabled: if dynamic velocity is enabled, then the program does not send an initial note-on until the key pressure exceeds the threshold for two keyboard scans. The difference between the pressure readings of those two scans is used to infer the velocity.

If tuning tables are enabled then we don't use pitch bend for microtuning at all – instead we search for the closest note to the one we want to play in our tuning table and send a note-on for *that* note.

Other Hardware

There are a number of other hardware components connected to the microcontroller. There is a Texas Instruments PCM5102a i2s audio DAC with a stereo line-out jack. There is a SN65HVD230 CAN bus transceiver, also from TI. There is a pair of MIDI ports connected to serial IO pins on the microcontroller. (The MIDI-in port is optically-isolated, as required by the spec.)

The display is a 240x320 touch screen with an ILI9341 controller, though we don't use the touch screen capabilities. Data is sent to the screen over a SPI bus. (The SPI clock is shared with the yellow LED on the Teensy 4.0, which is why the LED blinks when the screen redraws.)

Troubleshooting

Symptom	Cause	Fix
Mechanical		
continuous stuck note from synthesizer	lost or garbled MIDI messages	press NOTES OFF menu button to send a note-off command for all midi notes across all channels
	cable disconnected while note was playing	same as above after reconnecting keyboard with synth, or reset synth
key or menu button stuck on	FSR changes resistivity when in unusually warm environments	move keyboard away from heaters or direct sunlight
	overtightened screw	loosen screws near affected key or button, see if that resolves issue
	foreign debris on keyboard circuit board beneath force-sensitive resistor material	remove back plate and wooden front, including keybed, remove affected force sensitive resistor, clean both and reassemble
key or menu button has low sensitivity	fingerprint or oxidation on printed circuit board traces underneath force-sensitive resistor material	same as above, but scrub affected traces with isoproyl alcohol
knob stuck at full value	broken solder trace	disassemble keyboard, reflow solder joints on affected pot, and reassemble or contact support
micro usb port on microcontroller broken	mechanical stresses on usb cable or wear and tear	replace microcontroller (Teensy 4.0 with pins from pjrc.com) with current Mosaichord firmware loaded or contact support

light blinks on microcontroller when screen is redrawn	it just does that (Teensy 4.0 LED shares a pin with screen SPI clock)	it's normal
wooden key top comes loose from felt backing	wear and tear, or poor glue joint between wood and felt	apply thin layer of white glue to underside of key, place key back on felt with heavy object on top, wait for glue to dry (if key is lost, contact support for replacement)

MPE / MIDI		
USB synthesizer not receiving MIDI commands	USB MIDI output is disabled	enable USB MIDI under SETTINGS→OUTPUT→USB MIDI (enabled by default) or load device preset for a USB synth
MIDI synthesizer not receiving MIDI commands over DIN-5 MIDI cable	din5 MIDI output is disabled	enable din5 MIDI under SETTINGS→OUTPUT→DIN5 MIDI (disabled by default) or load a device preset for a non-USB hardware synth
only one every 15 or 16 notes sounds	MIDI synthesizer not in multi-timbral mode, only listening on one channel	consult synthesizer manual for instructions to put synthesizer in multi-timbral operating mode
	MIDI synthesizer does not have multi-timbral capability	mono-timbral synths aren't generally supported by the Mosaichord keyboard
	MIDI synthesizer is a mono synth	use an appropriate mono-synth device preset, or set the number of MIDI channels to 1 under SETTINGS→OUTPUT→CHANNELS.
erratic changes in the sound of notes	not all channels are set to same patch	change patch preset from Mosaichord under PATCHES→PATCH, verify that the synth has loaded the correct patch on all channels
	not all channels have same settings	reconfigure the synth to have all parameters the same for all channels, save configuration as a preset if necessary, (reloading all the patches or resetting the synth may also work)

tuning seems slightly off (sounds almost like 12-EDO)	pitch bend range on the synth is much narrower than the Mosaichord thinks it is	set pitch bend range on Mosaichord to match synth under SETTINGS→OUTPUT→BEND RANGE or reconfigure synth to match Mosaichord's bend range configuration
	effects or synth settings are obscuring sense of definite pitch	try disabling oscillator detune, chorus, vibrato, or tremolo effects
tuning is pretty far off	pitch bend range on the synth is much wider than the Mosaichord thinks it is	set pitch bend range on Mosaichord to match synth or reconfigure synth to match Mosaichord
	tuning table-based preset in use, but Mosaichord and synth don't agree on tuning table	set correct device preset for synth, and set tuning table manually on synth if necessary
notes cut out too quickly, have awkward-sounding envelopes	note-on attack and key pressure volume may be interfering with each other	turn off either pressure or velocity under SETTINGS→CONTROLS
reverb cuts off too soon	some synths apply reverb *before* volume control (CC 7 usually), which we often use to control per-note volume expression	not much we can do about this directly, other than to not modulate CC 7 with key pressure, or disable reverb entirely on the synth and route its output through an external reverb unit
noticeable key pressure lag	controller may be sending MIDI messages too fast for synthesizer to process	increase SETTINGS→CONTROLS→P BACKOFF until latency goes away ("pressure backoff" is the time interval between sending pressure updates in microseconds).

MPE synth does not recognize Mosaichord as an MPE device	synth may have missed the mpe initialization that happens when keyboard is booted or mpe device preset is loaded	re-send mpe initialization (SETTINGS→OUTPUT→MPE INIT) or re-load one of the MPE device presets
Seldom-Used Features		
unable to send/receive CAN bus messages	incorrect cable termination	verify 120 ohm resistor at both ends of CAN bus, turn on CAN bus resistor switch if needed
	no software handling	add desired software CAN messaging support to firmware source code, recompile and load