Manuel Odendahl

Algorithmic Music in a Box

Introduction

Since its conception in the early eighties, the MIDI bus has taken the world of electronic music by storm, and despite its limitations still remains an essential part of today's studios. The main attraction of the MIDI protocol for creative manipulation is that it represents symbolic information (notes, events, parameter manipulation, tempo information), rather than encoded audio. This allows us to use lightweight algorithms which can easily be implemented on small processors, and thus be used in DIY hardware. This paper does not go into specific implementation details, but provides an overview of MIDI DIY hacking.

There is a myriad of ways to use MIDI in a creative hacking context. MIDI often provides a way to poke into the internals of hardware synthesizer, so that their functionality, which is often cast in stone, can be extended by a clever outboard device. It also allows to create controllers that are customized for a certain kind of hardware synthesizer or performance approach, providing new ways of performing music and interacting with devices. Finally, custom music writing and generation tools can be built, allowing us to create algorithmically generated music that is tightly coupled to a custom hardware interface.

The purpose of this paper is to show in what ways a simple programmable MIDI controller with just a few basic controls (for example knobs, buttons or a joystick) can be used to completely change the workflow we have with existing MIDI hardware.

The MIDI Protocol

The MIDI protocol is a simple serial protocol implemented on top of a serial current-loop hardware bus. Data is sent at 31250 bps over a current loop, using a start bit and a stop bit for signaling, and 8 data bits. An optocoupler has to be used to receive MIDI information, while sending is done by simply toggling an output pin, sourcing 5 mA. The current loop is there to avoid ground loops when connecting multiple audio devices using MIDI. This means that interfacing standard microcontrollers to MIDI is very easy, as their UART component (most microcontrollers have one) can be used.

The MIDI protocol itself is very simple, and yet pretty cleverly built. The MSB of each byte is used to signal if the byte is a data byte or a status byte. Status bytes indicate the start of a command, which can be NOTE ON (0x90), NOTE OFF (0x90), CONTROLLER CHANGE (0xB0) and some more (the status bytes listed are those of interest to us in this article). The lower nibble of the status byte indicates on which "channel" the command is sent. Thus, each MIDI connection provides 16 distinct MIDI channels, which can be used to control different synthesizers for example.

Each status byte is usually followed by a few more data bytes (no MSB set) providing the actual information. For example, NOTE ON and NOTE OFF are followed by the note number and the velocity of the note. CONTROLLER CHANGE (which is used for example to send knob tweaking information) is followed by the controller number and the controller value. Most MIDI parameters thus have a resolution that goes from 0 to 127. There are quite a few extensions that allow for finer-grained parameters to be sent (NRPN), that allow relative parameters to be sent (relative CCs), as well as reduce the amount of data sent (running status), but in order not to burden the paper with technical detail, we will focus on just NOTE messages and CONTROLLER messages.

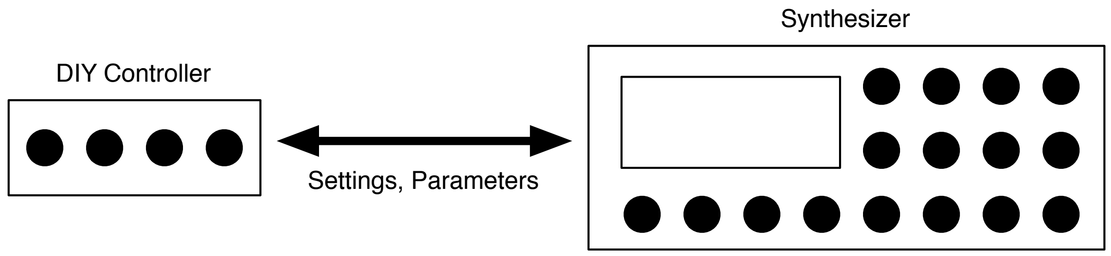
MIDI also has another class of messages called SYSTEM messages, which relate to all channels. These messages are comprised of SYSTEM REALTIME messages, which deliver clocking information, and SYSTEM EXCLUSIVE messages, which are pretty much a generic wrapper around binary data. MIDI CLOCK messages are a subclass of SYSTEM REALTIME messages. A clock event is sent at every 96th of the current tempo, allowing MIDI devices to synchronize to the tempo master. Additional SONG TRANSPORT messages allow the master to start/stop/position slaves inside a song structure. We will focus on tempo information in the "Tempo, Looping and MIDI Synchronization" part of this paper.

SYSTEM EXCLUSIVE messages are used by various manufacturers to control different manufacturer-specific of their devices. For example accessing more complicated parameters that can not be controlled directly using CONTROLLER CHANGE messages, storing and loading patches, polling the current user interface status and much more. We will explore SYSEX MESSAGES and what is possible with them in the "Hacking the Synthesizer User Interface" part of this paper.

Hacking the Synthesizer User Interface

This chapter shows a few ways in which the normal user interface of a synthesizer or hardware MIDI device can be extended through an external device, often completely changing the way in which the synthesizer can be used, often turning it into a completely different instrument.

Usually, SYSEX messages is the easiest way of accessing the internals of a synthesizer. One can often modify internal sound generation parameters that are not accessible through CONTROLLER CHANGE messages. A good example of this is the Elektron Machinedrum, which is the hardware synthesizer we have been focusing on. The MachineDrum provides 4 internal effects: Delay, Reverb, Equalization and Compression. However, the parameters for these effects can only be accessed using an internal menu, which takes about 5 key hits to access, and blocks the access to other parameters when active. Thus, tweaking the internal effects while performing is quite tricky and dangerous, and thus never done. However, the parameters can be accessed with SYSEX messages. We built a small controller that allows the user to directly control the parameters of the internal effects, which led to a completely new way of using the machine. With the controller, we were able to control the Delay Feedback parameter, allowing it to go into the dangerous feedback region knowing we were able to tame it at all times. This completely changed the approach we had to the machine, and paved the way for all the MIDI work we did later on.



Another way we use SYSEX messages with the MachineDrum and MonoMachine synthesizers is loading the saved sound parameters (called a Kit), and analyzing them. The MachineDrum is a drum machine providing a plethora of drum and percussion sounds (each one of these called a Machine). Some of these sounds are pitched, but not in a conventional musical way (not following notes). Accessing the SYSEX information, parsing the Kit settings, we were able to map conventional MIDI notes to the different pitch settings of the active Machines. This allows us to play a polyphonic snare drum using the keyboard, again completely changing the approach we have to the machine, as it is suddenly not a drum machine anymore, but a powerful and weird polyphonic synthesizer.

Being able to completely program our own controller allows for a host of tricks. For example, specific key combinations can be used to vary multiple parameters simultaneously (see also "Morphing Parameters and Exploring Parameter Spaces"), often driving sounds into very weird sonic regions. Accessing the saved settings over SYSEX, we are able to "restore" the original settings by just pressing a single button (something which would often require a few menu items on the synthesizer itself), allowing us to be very bold in our sound modifications.

Of course, much more prosaic things are possible through SYSEX access: archiving settings and patterns, merging patterns, storing sounds with useful names to be loaded later on (which is difficult to do on most hardware synthesizers), morphing between stored settings. A very nice feature is for example periodically storing the current pattern and current settings of a synthesizer while working on a song, and then later on allowing for a "Time Machine" kind of access. Undo memory is usually very limited or inexistant on a hardware synthesizer, sometimes making work very painful.

Morphing Parameters and Exploring Parameter Spaces

The number of parameters on synthesizers are usually pretty big, and even with a restricted number of parameters, the synthesis algorithms are pretty complex, allowing for a big range of different sounds to be created with a few basic ingredients. When adding complex modulation possibilities like LFOs, cross-synchronization of oscillators, multiple envelopes, the sound design space becomes virtually illimited. A lot of work has been done to come up with new ways of exploring the sonic spaces, for example the work of Palle Dahlstedt (http://www.ituniv.se/~palle/pmwiki/pmwiki.php).

A simple way to implement this kind of workflow on our devices is to make the controller do randomization of synthesizer parameters. For example, turning a knob on the controller can change a number of parameters inside a specific range, allowing the user to explore different combinations quickly without having to tweak every parameter by himself. This kind of modification becomes even more fun by using a different kind of hardware controller such as a joystick, or an acceleration sensor.

Modifying multiple parameters at once in a randomized fashion can however quickly lead to sonic chaos. A good way to tame the chaos is to use genetic mutation. A sound seed is planted and then evolved into different directions using the hardware controller. Once an interesting combination has been found, this can be made into a new seed and evolved again, or merged with another number of stored seeds. The VST Synthplant by Sonic Charge (http://www.soniccharge.com) or the genetic mutator of the Clavia G2 Modular Synthesizer work that way. Adding this kind of randomization to existing hardware synthesizers, maybe coupled to clever parameter mapping as described in the first part of this paper, breathes new life into old machines.

Tempo, Looping and MIDI Synchronization

Most music, and especially electronic music has a firm rhythmic foundation. This is also visible in the MIDI specification, providing SONG START, STOP and MIDI CLOCK messages. The MIDI CLOCK message (0xF8) is periodically sent on every 96th note, and has precedence over every other message, thus allowing clock synchronization to be tight. The receiving side can either clock its internal clock directly from the MIDI CLOCK messages, or use a phase-locked loop to generate its own clock. Once two devices are synchronized, they run at the same tempo, and the slave devices follows the tempo of the master device.

Implementing tempo synchronization in our devices allows us to take the user interface modification to another level, as we can now synchronize our modifications to the current tempo. This allows us for example to loop parameter modification for a rhythmically sound duration (for example 4 bars). It also allows us to cue parameter modifications to become active on a specific beat.

Having a synchronized clock also allows us to generate LFOs for specific parameters. An LFO is a low-frequency oscillator that slowly changes a parameter according to different waveforms. For example, it can make a parameter go up and down every 2 seconds over a range of 40 units. A clock synchronized LFO allows us to vary sounds in a rhythmically interesting manner (for example allowing for polyrhythms on a sonic level). The standard way to provide this kind of parameter manipulation is to use the automation features of a sequencing program on the computer, and either record knob tweakings or draw them into a timeline. This is quite time-intensive and doesn't allow for the kind of exploration that is easily done on hardware.

Often, hardware synthesizer only provide a limited number of LFOs, or with limited waveform possibilities. Adding our own external LFOs allows us to extend the possibilities of the hardware synthesizer while keeping the amount of work involved low. It also allows us to explore the possibilities of an LFO in itself, for example: replacing the waveform with a table of values, synchronizing two LFOs together, locking the LFO values to interesting values (for example tonality dependent, or rhythmically sound).

Another example combining the tempo synchronization and the parameter hacking is a hack for the Elektron MachineDrum. The MachineDrum allows to load and record samples, and move their start- and endpoints parameters (where the sample starts and where it ends). Assuming we have loaded a rythmic sample (for example a drumloop), moving the playback points in units of 8 allows us to start the sample at different rythmically sound values, thus "chopping" the sample, also allowing for interesting reverse sample effects and pitchdown/pitchup effects. This kind of manipulation is the basis for a lot of electronic music, for example drum'n'bass, breakcore, IDM. However, the standard interface of the MachineDrum makes it impossible to tweak the playback points in this way, and also to modify the parameters on rythmic hits. Using the external controller, we are able to move the endpoints and startpoints in rhythmically sound values (units of 8 for 16th notes, 16 for 8th notes, etc...), and modify the parameters only on rhythmic events and not in between. This allows for tight live drumloop chopping. We can take the concept further and implement automatic drumloop-chopping algorithms such as those invented by Nick Collins (http://www.cogs.susx.ac.uk/users/nc81/).

Algorithmic Rhythms

Taking tempo synchronization further, we can actually implement complete sequencers on our DIY hardware. A simple kind of sequencer that lends itself well to a hardware interface is an arpeggiator. An arpeggiator takes a number of input notes (a chord), and plays back notes from this chord synchronized to a clock, thus turning the chord into a quick arpeggio. Mapping parameters of this arpeggiation to hardware controls allows to create interesting rhythmic and harmonic texture, playing with note length, note range, and different other parameters. For example, something "common" arpeggiators often don't provide is to couple synthesis parameters to the arpeggiation. This can be easily added to our external controller, allowing us to play with accents, with rhythmic modification of synthesis parameters, once again turning our external controller into a completely new instrument of its own.

We can also devise new sequencer ideas. A very fascinating algorithm to generate rhythms is the euclidean algorithm, which can be used to generate most of the western and african rhythms (see the paper The Euclidean Algorithm Generates Traditional Musical Rhythms by Godfried Toussaint). Using just 3 numbers (number of hits, length of pattern and start offset), we can generate complex and natural sounding polyrhythms. Implementing this algorithm on our hardware controller, we can very quickly and easily modify and explore different rhythms, without having to actually calculate the rhythms and input them into a normal sequencer.

Of course, this can be extended to any musical theory that lends itself well to an algorithmic computer-based implementation. Having direct haptic access (through the controller interface) to the sequencing parameters allows for a much more indepth and natural exploration of different kind of algorithmic musical processes.

Conclusion

We hope that we were able to show the immense range of possibilities that a DIY MIDI controller offers in terms of musical exploration. It allows us to extend (hack) existing MIDI devices, adding custom functionality. It allows us to explore musical possibilities through an intelligent and customized interface. It extends the performance possibilities by allowing rich tempo-synchronization possibilities, and finally permits us to implement, prototype and interact with complex algorithmic sequencing models. Videos of these features as well as schematics and sourcecode can be found on our website at http://ruinwesen.com/ .