MODULE-LEVEL MIDI

Sean LUKE[1], **John TUFFEN**[2], and **Mathias BRÜSSEL**[3]

[1]*Department of Computer Science*, **George Mason University**, Washington, DC, USA
[2]**wonkystuff**, York, UK
[3]**tangible waves**, Murnau, Germany

ABSTRACT

We describe a set of conventions to run MIDI over the cables between individual modules in a modular synthesizer. Our approach allows modules to modulate one another; or to be collectively controlled by the musician; or both. The method deals with a variety of special problems arising from this fine-grained use of MIDI, such as module polyphony, MPE incorporation, parameter spaces, and avoiding full graphs and MIDI merge. We detail and discuss the issues and challenges, then offer a specification and provide usage examples.

1. INTRODUCTION

Module-level MIDI implements MIDI 1.0 [1] at the per-module level in a modular synthesizer. We envision MIDI as just another cable signal, integrated with gate / control voltage or audio. The goal is to create a flexible way for modules to control one another, to be collectively controlled by the musician, or both. Modules send MIDI to one another over standard modular cables rather than a bus, though a bus is not precluded.

Why would one wish to contaminate modular with MIDI? First, MIDI greatly simplifies the wiring for *polyphony* and *multitimbralism* in a modular synthesizer, and enables the use of *MIDI Polyphonic Expression*. Second, it helps automate the control of modules with *large numbers of parameters*. Third, it enables *patch storage and recall*. Fourth, it provides stable *pitch control* and a pitch standard with *microtuning*. Fifth, it provides *compatibility* with the external world.

In this paper we present three major challenges to implementing module-level MIDI, and how we overcome them. We then present conventions for control of modules by one another, or by an external controller. Our goal is to both provide foundations to be used by modules *now*, and a roadmap for more elaborate applications in the future. This roadmap can be modified as necessary later. Thus the spec is only version 0.5.

2. RELATED WORK

Modular synthesizers, as originally proposed by Harold Bode [2], consist of multiple hardware *modules* that create or modify audio, or produce analog gate or *control voltage* (CV) to modulate other modules' settings. Audio or gate/CV flow from module to module via *patch cables*.

MIDI, or other forms of digital control of modules, used to be anathema in the modular world, but over the last few years it has seen significant growth in interest. The most common use of MIDI in a modular setup is modules that translate 5-pin DIN or TRS MIDI[1] into gate/CV for other modules. This configuration is meant to allow a musician, via an external controller, sequencer, or DAW, to send notes and parameter changes to select modules in real time.

Digital control of or among modules in modular synthesizers has to date largely used internal bus protocols. The forerunner in this area has been Buchla, whose systems feature an internal bus with multiple virtual lines meant to allow modules to send MIDI to one another or to receive MIDI from an external controller [3].

MIDI has also been run in a limited fashion on top of the internal gate/CV bus in Doepfer's Eurorack modular format. This scheme, developed by a group of Eurorack developers, is known as *Select Bus*, and has been primarily used to broadcast Program Change (PC) messages to modules to change their presets [4]. It is incompatible with other manufacturers' use of the gate/CV bus, including Doepfer's. Expert Sleepers has also introduced modules (such as the CVM-8 [5]) with one or more separate, dedicated internal Transistor-Transistor Logic (TTL) serial MIDI bus connectors.

Eurorack has also seen *I2C* used as an alternative bus transport. I2C is a two-wire packet-oriented communication scheme common in electronics. As popularized by the Monome Teletype [6], a simple control protocol runs on top of I2C from a leader module to some number of downstream follower modules. The protocol runs at 100kbps. Unfortunately it has not been standardized, resulting in a plethora of competing I2C-based messages among the modules which have adopted it. Some Buchla designs also employ I2C.

Most bus designs assume a single leader per bus. This means that a single bus line cannot easily support the scenario where module A controls modules B and C while B also controls C. To realize a general graph structure of control would require many real or virtual bus lines (as

[1] While TRS MIDI uses similar cables, it is not compatible with Eurorack as it requires specific circuitry, including an optoisolator.

in Buchla) and complex addressing. More elaborate bus protocols, essentially network protocols such as CANBus that define full graphs, have been occasionally proposed but not commonly adopted.

Beyond busses, other digital control schemes have focused on easing the use of polyphonic modules. Notably, TipTop has recently introduced *ART* [7], an event protocol running at 1.25 MBPS over standard cables in Eurorack. ART is accompanied by *Polytip*, a format for running parallel analog signals (importantly audio) using a cable with a plug identical to USB-C, but not using the USB-C protocol.

MIDI is also often used to interconnect and control individual guitar pedals on a pedalboard: this is somewhat analogous to modules in a synthesizer. Many guitar pedals accept MIDI, and certain pedals (an example would be those from Morningstar [8]) are meant to distribute PC or CC messages to them.

3. MIDI OVER MODULAR CABLING

In 2022 wonkystuff [9] introduced MIDI running as TTL serial over the 0.1" Dupont cables used to connect modules in the AE Modular synthesizer format [10] alongside gate/CV and audio. This scheme employs a channel distribution module to break TRS MIDI into separate per-channel message streams, each of which can then be routed to chains of MIDI-controlled oscillators, triggers, and gate/CV generators. A single chain of modules could be treated as a one monophonic synthesizer; and parallel chains could collectively implement polyphony via MPE [11] and other ways.

AE Modular makes this particularly easy because it is strictly 0–5V with a common ground, and standard MIDI is likewise also nominally 0–5V. AE Modular cables are single wires with ground on the power bus, and so are sufficient to run MIDI as TTL serial data.

wonkystuff's approach is primarily meant to allow the musician to control multiple modules using a single external controller, sequencer, or DAW. However it has certain limitations. First, it does not have any way to distinguish among the modules listening in on the same channel. This is particularly problematic if one wanted, for example, three oscillators on a single channel but with different parameter settings, such as detune. Second, it does not allow modules to *modulate one another*: for example, a MIDI-controlled envelope module modulating a MIDI-controlled oscillator module, while both are controlled by a DAW. Third, it does not address the issues raised by MPE, nor MIDI-controlled polyphonic modules, a popular recent trend.

In this paper we significantly extend this approach, proposing a set of MIDI conventions to follow in order to address these and other issues, while staying compatible with MIDI standards. We have expended effort in making basic forms of the protocol easy for simple devices to implement, while providing more capabilities to more sophisticated modules and systems.

3.0.1 Why Not Just a Bus?

We are interested in running MIDI over cables, rather than a bus, because it makes it much easier to do complex topologies, and because it does not require memorization of virtual "behind the scenes" routings. It does, of course, require more visible wires. We must admit that because of these very wires, MIDI over cables is more tangible and retains much of modular's physical charm: it's just another cable routing like CV or audio.

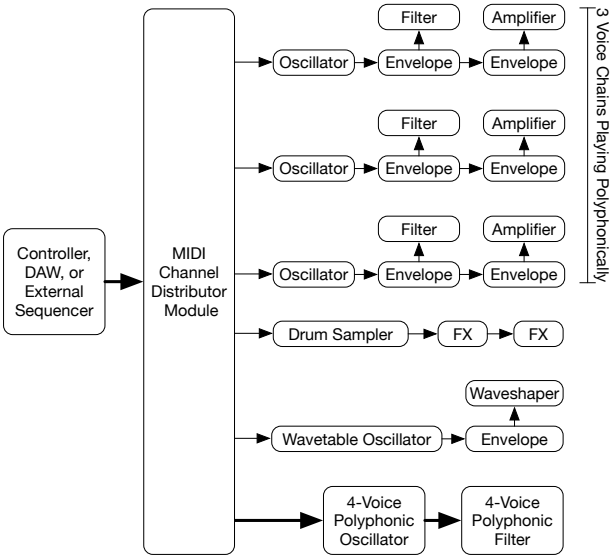

Figure 1. Over-Elaborate MIDI Connectivity Tree Example. The Distributor module breaks out MIDI data by channel to individual voice chains. Thick arrows indicate single MIDI connections containing multiple channel voice data. Note the three identical polyphonic voice chains, the separate wavetable voice chain, and the *two* FX modules in the Drum Sampler chain. Finally, note that the envelopes are organized as non-leaf nodes, to enable optional MIDI Injection.

That being said, a bus is not in any way precluded: indeed we have discussed the possibility of combining cabling with a MIDI bus internally on AE Modular.

3.0.2 Why MIDI 1.0?

MIDI 1.0 is now 40 years old, and its failings are very well understood. But its advantages are also compelling. MIDI has extensive tooling available, is flexible and extensible, and is a simple and easily parsed serial protocol. But most importantly, it is omnipresent in the music industry. By adopting conventions layered over MIDI, we immediately gain access to a huge number of devices and software.

Why not MIDI 2.0? MIDI 2.0 addresses MIDI 1.0's primary flaws, but it is a large and complex protocol that may be difficult, if not impossible, for small processors to implement. It is not yet widely adopted in the industry, and is immature. It would seem to be a poor option for modular systems with simple modules.

Last, proprietary protocols running on top of I2C have seen increasing popularity in certain Eurorack circles; but they are neither standard nor standardized, and often are little more than arbitrary MIDI translations.

4. CHALLENGES

We begin with three challenges in digital control of modules via MIDI, and how we will address them. The first challenge is **topology**. MIDI is meant for one controller to control multiple downstream devices. We want to allow modules to control one another, but we want to avoid a full graph structure and MIDI Merge. The second challenge regards **namespaces** and **voices**. MIDI has a single namespace: channels. But when issuing parameterized control, we need to distinguish between receiving modules based both on the voice they are assigned to and the particular module in that voice. We also consider polyphony and polyphonic modules. The third challenge lies in making certain that the changes we make in the first two challenges do *not* require a monolithic, complex protocol that cannot be implemented by **simple devices**: we need a way to ramp up in complexity as needed.

4.1 Challenge 1: Topology and Speed

Anyone who has connected many hardware synthesizers is painfully familiar with the complexity of wiring them. The large number of modules in a modular synthesizer can make this even uglier. Ideally we would place all the modules on a network and allow any module to communicate with any other, though this would be complex to implement, require memorization, and lose the immediacy and (in truth) the charm of modular.

Regardless, MIDI is far from a network: it is a unidirectional, daisy-chained communications protocol. We abandon a bus and allow MIDI to assume a tree topology, with a leader at the root sending messages to all nodes in the tree. The root is likely a controller, DAW, or hardware sequencer, under the control of the musician. Non-leaf nodes in the tree both respond to and forward these messages to their children. Figure 1 illustrates an (overly elaborate) example of a tree of MIDI control among modules.

In a full graph, allowing any node to talk with any other node would require MIDI Merge. We permit MIDI Merge but actively avoid requiring it due to its complexity and introduction of cycles. But we can claw back some of this power by allowing modules to modulate nodes in their subtrees. We do this by optionally introducing a "lightweight merge" in the form of *MIDI injection*.

Injection is simple: before it passes data from its parent to its children, a module is permitted to *inject* some MIDI messages into the stream to send to one or more of its children. This would allow Module A to modulate Module B, and B to modulate C, while all of them are being controlled by the root. Generally non-leaf nodes would often be modulators (envelopes, say), and modulation targets would be leaf nodes. Figure 2 illustrates MIDI injection.

MIDI is also slow: it runs at 31250 bits per second. [2] But assuming MIDI does not exit the modular synthesizer, there is no reason why it couldn't run as fast as we would like internally. 115200 BPS seems reasonable as it is a common baud rate. Speed would be constrained by the maximum speed afforded by the medium (patch cables) and the maximum rate that the basic modules' microcontrollers can handle.

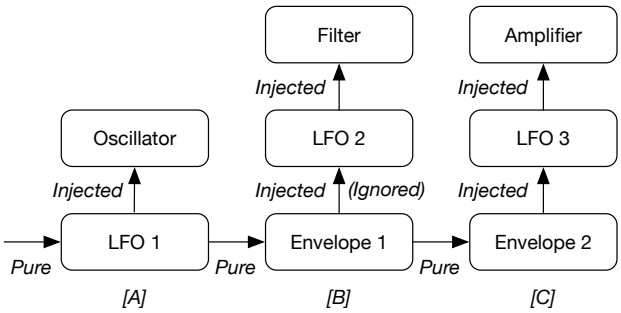

Figure 2. Three Injection Examples. [A] LFO 1 injects modulation data into the stream to control the Oscillator pitch, while passing "pure" non-injected data to Envelope 1. [2] Both Envelope 1 and LFO 2 modulate the Filter. This is done by Envelope 1 injecting modulation data to LFO 2, which ignores it and passes it on while also injecting its own data. Envelope 1 passes "pure" data to Envelope 2. [C] Envelope 2 injects data to modulate LFO 3's depth. LFO 3 responds to this modulation data, and removes it from the stream before injecting its own data to modulate the Amplifier.

MIDI traditionally incurs latency as data is transferred from module to module. Part of this is due to its use of optoisolators (which we do not need). But excess MIDI injection could also have significant latency costs.

Last, bidirectionality is useful for transferring and storing presets. While a MIDI tree is unidirectional, there's nothing stopping modules from providing a single back-channel wire to provide bidirectionality, at least between two modules.

4.2 Challenge 2: Voices and Namespaces

If we are to control parameters of multiple modules, we will have a namespace problem.

Historically MIDI has had just one limited namespace to distinguish between devices: its 16 MIDI channels. However the recently adopted MIDI Polyphonic Expression (MPE) specification also uses this namespace to distinguish among polyphonic voices in the same device, and has introduced "MPE Zones" in a halfhearted attempt to allow the two usages to coexist [11].

Inside a modular synthesizer we are faced with a bigger namespace issue. First, as polyphony and multitimbrality are now popular in modular synthesizers, we must distinguish among voices. Second, we must distinguish between different modules in a given voice: an oscillator will have different parameters than a filter.

To unravel this, we suggest adopting the approach suggested by MPE: commonly each channel is a separate voice, [3] and the subtree of modules which collectively pro-

[2] You may be interested as to why. Early MIDI synthesizers had CPUs operating at multiples of 1 MHz. To quote: "The 31.25 kHz clock can also be obtained from hardware, for example, by dividing 1 MHz by 32." ([12], page 7, §"Hardware").

[3] There are exceptions. For example, a drum module would likely produce different drum sounds per-note on the same channel.

duce that voice are known as a *voice chain*. Within a voice chain, modules may be distinguished from one another by their *ID*, a value ≥ 1. As all modules in a given voice chain will be listening in on the same channel, IDs are then used to divide up the parameter space in CC and NRPN. Figure 1 illustrates the need for this namespace, as it has polyphonic voice chains, multiple envelopes per chain, and multiple effects units.

A module might set its ID via a DIP switch, or via a value stored in EEPROM, and very trivial modules might be simply fixed to an ID, though this is discouraged. NRPN is straightforwardly partitioned according to ID. But CC is very limited, and so we have attempted to divide it optionally and flexibly by ID. Modules are encouraged to use certain regions in CC according to their ID but we allow much leeway. If necessary, and if NRPN is not feasible, a module may allocate additional IDs for more CC space. We have also provided a "poor-man's NRPN", called *auxiliary parameters*, for IDs without CC allocations (Section 5.5).

In some cases, modules need both separate parameters and joint parameters. For example, two oscillators might wish to share the Pulse Width parameter but have different Detune parameters. A module is permitted to share portions of its ID parameter space with another.

4.2.1 Polyphony and Polyphonic Modules

Several models of polyphony are achievable. First, there is standard MIDI 1.0 *one-channel polyphony,* where a *voice assigner* module would distribute note messages to individual voices, as well as copying voice-parameter CCs to all voices. This could also be encapsulated as a single *polyphonic module*. Second, a *polychain* is a string of monophonic oscillators, each of which passes note data along to a subsequent oscillator via soft MIDI THRU when it is already playing a note. Polyphony in this case would likely be limited by latency. Third, in *MPE* each note is pre-distributed to its own channel by a controller. The distributor would then send each channel to a separate voice chain, or would group several channels (hence voices) to send to a polyphonic module.

A polyphonic module may be thought of as several monophonic modules glued together, each in its own voice chain. A polyphonic module could sport one MIDI input per voice, or a single MIDI input for multiple notes along a single channel, or one input for multi-channel data. In the second case, it might terminate multiple monophonic voice chains (as a multi-VCA package, for example). See Figure 1 for an example of polyphonic modules in a MIDI tree configuration.

4.2.2 Why not Distinguish Modules by Channel?

There are many reasons not to assign each module in a voice chain to its own channel, not the least of which is incompatibility with MPE. But one issue is that notes are sent to specific channels. In a modular synthesizer several modules may need to respond to the same note, such as oscillators responding to pitch, envelopes or LFOs responding to gate, etc. But they cannot do this if they are on different channels. Channels need to be associated with voices.

4.3 Challenge 3: Simplicity versus Versatility

We imagine that for most people, needs will be very simple, and many modules will have small microcontrollers. Thus we need a protocol that can be both simple or more sophisticated and versatile depending on need and cost. We address this in five primary ways.

First, we assume that MIDI voice data for a given voice chain has been filtered to a single channel by a distribution module, and thus most modules in a chain may operate as OMNI. Polyphonic modules may instead benefit from MIDI sent with multiple channels.

Second, modules have default ID settings and these defaults vary by module type (oscillator, filter, etc.) Thus modules often can be simply plugged in and played in basic configurations. Trivial modules need not have a user-settable ID at all (though this is discouraged) if there are a small number of them and their default IDs are all different.

Third, simple devices need only respond to basic 7-bit CC parameters, but 14-bit CC and NRPN are optionally provided for more elaborate parameters. CC parameter regions are set up to be flexible and not set in stone.

Fourth, many systems will just be controlled externally, and not need module-to-module MIDI modulation: and so basic modules could ignore this facility. But module-to-module MIDI modulation is made to be easy: modulators need only be set to modulate one of 8 generic parameters that we have set aside in the CC space. Modulation targets would just map these to their first 8 CC, Auxiliary, or NRPN parameters.

Fifth, the use of MIDI does not preclude traditional CV in any way: they can be mixed without incident.

Along these lines, we can think of the protocol at different levels of capability, depending on need:

4.3.1 Level 1: Trivial Support and CC Learn

At its simplest, the protocol would support a voice chain for a monophonic synthesizer with a small number of modules and little external control by the user. An upstream MIDI distribution module would have filtered out undesired channels, so a downstream module can use OMNI. Such modules would have hard-coded IDs and only use the CC parameters allotted to those IDs. If two modules had conflicting hard-coded IDs, one could still have its CCs remapped to elsewhere in the CC space via a CC Learn function or a sysex message. Having hard-coded IDs is discouraged however.

A module would provide only "pure" (hardware) MIDI THRU. Polyphonic MPE-style voice chains would be handled entirely by the MIDI distribution module with multiple per-channel MIDI OUTs.

4.3.2 Level 3: Settable NRPN and Program Change

More advanced modules would have settable IDs, particularly if they wished to use the additional parameter space afforded by Auxiliary parameters or NRPN. These modules might also support presets via Program Change, and saving presets via Program Save (Section 5.5). Note that Program Change would affect *all modules* on a channel/voice chain that respond to it. An optional proposed feature, *Module*

Select (Section 5.4), would allow one to to indicate which modules should respond to a Program Change/Save message.

4.3.3 Level 3: Modulation Modules

Some modulators (envelopes, LFOs, sequencers, etc.) could change *other* modules' parameters via MIDI. These modules could simply change one of eight generic *modulation CCs*, and their modulation targets would map these CCs to 8 parameters of their choice. Modulators would perform *MIDI injection* to, and removal of modulation messages from, the MIDI stream.

4.3.4 Level 4: Polyphonic Modules and Merge

Polyphonic modules would support a single multi-channel MIDI connection to connect to other polyphonic modules, or optionally support multiple OMNI MIDI connections for simple voice chains. We might also see a MIDI Merge module if useful.

5. PROPOSED CONVENTIONS (V. 0.5)

This is v. 0.5, and is subject to further revisions.

The topology of the system is a tree. The root node is often a musician's controller, DAW, or external sequencer. Most remaining nodes are individual modules. Parent nodes in the tree are normally directly connected to child nodes via user-pluggable cables, though they could be connected via a bus. Data flows from parent to child only. We adopt the MIDI THRU concept: nodes can pass data from their parent on to their children.

Nodes normally pass data through a hardware MIDI THRU. Nodes wishing to modulate other nodes are also permitted to *inject* data into a *software* THRU, that is, add data to the stream passed to their children. Nodes may have multiple MIDI THRUs and they can be of different types with different kinds of injection: however they should always have at least one "pure" (non-injected) MIDI THRU. Nodes may remove data passed to them by their parent prior to injection of new data: see Section 5.3. See also Figure 2 for examples of injection and MIDI THRU.

A node may have multiple parents only if it keeps their incoming data streams separate (such as a polyphonic module), or if it fully implements MIDI Merge, a rare but permitted case. MIDI Merge plus MIDI THRU make cycles possible: cycles can overwhelm the MIDI stream and their impact on the topology is not defined. Nodes may to have additional MIDI connections outside the primary topology (such as for bidirectional patch transfer), but their effect is not defined here.

The speed of MIDI from module to module should be at least 31250 BPS, but can be faster than this if it is standardized as such on a given modular platform.

Nothing here forbids modules from responding to MIDI as they like, such taking control over one or more entire channels if they see fit. For example, it is entirely reasonable that a sophisticated polyphonic oscillator module might simply claim an entire channel and all note and CC events on that channel. However it is often the case that multiple modules must listen to the same channel and respond to notes and CCs in order to to work together to form a voice. They must do this without conflicting with one another. Other modules must control one another via MIDI while also being externally controlled themselves. The conventions below deal with cases such as these.

ID	*Type*
1	Oscillators, Samplers, Note→Gate/Trigger Modules
2	Envelopes
3	LFOs, Sequencers, Clock→Gate/Trigger Modules
4	Filters
5	VCAs, Mixers
6	Effects, Audio Processors
7	CC→CV Generators
8	Miscellaneous

Table 1. Default ID Types

5.1 MIDI Channels, Voices, and Voice Chains

Usually a MIDI channel is associated with a *voice*, that is, the set of modules which together produce a sound in response to a MIDI note. Some voices can have the same parameters and so collectively act polyphonically, while multitimbral voices may be distinct.

Commonly a distributor module receives MIDI from an external sequencer and breaks it out to MIDI outputs per-channel (thus per-voice). Each MIDI output is routed to a subtree of modules working together to produce one voice: this is a *voice chain.* As the channels have been broken out, modules in a voice chain may respond to MIDI voice messages in OMNI mode.

A *polyphonic module* may subsume multiple voices, hence multiple channels. Polyphonic modules may receive separate individual streams (as OMNI) and so participate in voice chains; or they may receive and send multiple channels via a single MIDI connection.

A distributor module receiving MPE ought to copy messages from the MPE master to the key channels as it deems appropriate.

5.2 Module IDs and Parameters

Each module in a voice (channel) has a unique ID 1–15, commonly 1–8. The ID determines the region of parameters allocated to the module in the CC, Auxiliary Parameters (Section 5.5), and NRPN spaces. Modules have default assignments based on the module type, shown in Table 1.

Modules are encouraged to have user-changeable IDs, but simple modules can have hard-coded IDs with default assignments under the (poor) assumption that the user only has one module of each type per voice. Indeed, if a module does not use CC, Auxiliary Parameters, nor NRPN, its ID is at present not relevant.

Module IDs 1–8 have parameters allocated as CCs. Each ID has two 14-bit CC pairs called "a" and "b", and five 7-bit CCs ("c" through "g") allocated. We have set some default ID types regions to include some classic CC names which show up in DAWs and controllers.

CC	*Name*	*CC*	*Name*	*CC*	*Name*	*CC*	*Name*	*CC*	*Name*	*CC*	*Name*
0/32	Bank Select	16/48	3a (a/i)	64	Sustain Pedal	80	3g	96	Increment	112	1g
1/33	Mod Wheel	17/49	3b (b/h)	65	1c Glide Switch	81	4c	97	Decrement	113	Mod. c
2/34	Open Breath Controller	18/50	4a (a/i)	66	1d	82	4d	98	NRPN LSB	114	Mod. d
3/35	Auxiliary	19/51	4b (b/h)	67	1e	83	4e	99	NRPN MSB	115	Mod. e
4/36	Open Foot Controller	20/52	5a (a/i)	68	Legato Switch	84	4f	100	RPN LSB	116	Mod. f
5/37	Glide MSB/Open LSB	21/53	5b (b/h)	69	1f	85	4g	101	RPN MSB	117	Mod. g
6/38	Data Entry	22/54	7a (a/i)	70	2c	86	5c	102	7c	118	Mod. h
7/39	Volume MSB/Open LSB	23/55	7b (b/h)	71	2d	87	5d	103	7d	119	Open
8/40	1a (a/i)	24/56	8a (a/i)	72	2e Release	88	5e Hi Res Vel	104	7e	120	All Sound Off
9/41	1b (b/h)	25/57	8b (b/h)	73	2f Attack	89	5f	105	7f	121	Reset Ctrlrs.
10/42	Pan MSB/Open LSB	26/58	Mod. a	74	MPE Timbre	90	5g	106	7g	122	(Reserved)
11/43	Expression Controller	27/59	Mod. b	75	2g Decay	91	6c FX1 Amt	107	8c	123	All Notes Off
12/44	6a (a/i) FX Ctrl 1	28/60	Open	76	3c Vib. Rate	92	6d FX2 Amt	108	8d	124	(Reserved)
13/45	6b (b/h) FX Ctrl 2	29/61	Open	77	3d Vib. Depth	93	6e FX3 Amt	109	8e	125	(Reserved)
14/46	2a (a/i)	30/62	Open	78	3e Vib. Delay	94	6f FX4 Amt	110	8f	126	(Reserved)
15/47	2b (b/h)	31/63	Open	79	3f	95	6g FX5 Amt	111	8g	127	(Reserved)

Table 2. CC Parameters. The first two columns are 14-bit CC pairs. Each module ID has two 14-bit CCs (a, b) and five 7-bit CCs (c...g). A 14-bit CC may be split into two 7-bit CCs: the parameter *a* may be split into *a* (MSB) and *i* (LSB), and *b* into *b* and *h,* hence the notation (a/i) and (b/h). Also shown are System, Open and (Reserved), and Auxiliary parameters, and Modulation CCs (a...h). ID default regions are set up to attempt to align with some CCs that have optional, traditional names. Open parameters may be used with care, with the understanding that other modules may also respond to them and that their purpose may change in the future. (Reserved) parameters may not be used.

A module is free to split one, or both, of its 14-bit CCs into 7-bit CCs if it needs 8 or 9 CCs. If just one is split, it should be "b", split into "b" (the former MSB) and "h" (the former LSB). If two are split, then "a" may also be split into "a" and "i" respectively. Table 2 shows the allocated regions for each ID.

Parameter space is allocated to IDs 9...15 as discussed in Section 5.5. Also, NRPN allocates 256 parameters to each ID, as shown in Table 3. The first 9 correspond to parameters "a" through "i" in the CC map allocated to the module. NRPN parameters are 14-bit even if their corresponding CC parameters are 7-bit. Reserved NRPN parameters should be ignored.

Modules may also change IDs and this ability is strongly encouraged. Some modules might plausibly use CC Learn (listening for an incoming CC and memorizing it) or other remapping scheme to relocate their CCs to regions normally allotted to other (missing) modules. However in this case they still ought to have user-changeable IDs if possible, or at least a fixed ID for default CC values.

A module may be assigned more than one ID if the need arises. Additionally, in some cases two or more modules may wish to share parameters in common in addition to having separate regions. For example, two oscillators may wish to both be changed by pulsewidth modulation, but have detune changed separately. This should be done by having one module set to respond to certain parameters of the other, ideally by ID.

A polyphonic module could have different IDs for each voice, but normally they would be the same ID.

A module might also perform all the functions of a voice and not need other modules. Such modules are effectively synthesizers in and of themselves and may be assigned a full channel per voice: they may disregard the suggested parameter assignments described here, and do as they wish with CCs and with NRPN, though they might wish to retain them for consistency.

NRPN Region	*Range*
0–255	[Reserved]
256–511	The first 256 parameters for ID 1
...	
3841–4097	The first 256 parameters for ID 15
4098–16383	[Reserved]

Table 3. Allocated NRPN Regions

5.3 Modulation CCs

Two 14-bit CC parameters and six 7-bit CC parameters are designated *Modulation a...h*. These generic parameters may be injected by modulation modules such as envelopes, LFOs, or sequencers and sent to their children. A child responds to these CCs as if they were its first eight CC or NRPN parameters (a..h). If a module does not implement that many parameters, it may ignore the remaining modulation parameters.

Modules are free to ignore this feature. Otherwise they are encouraged to have, as a switchable option, the ability to respond to or to ignore incoming modulation CCs. Modules that modulate other modules via MIDI injection *must* have this option. If a modulation module itself responds to a particular modulation CC, then it must remove that CC from its stream before injecting new messages. See Figure 2 for examples of injecting data and and ignoring the same.

The two 14-bit modulation CCs are 14-bit even if the corresponding regular CCs are 7-bit. They have special rules to improve atomicity in 14-bit CC. When an MSB arrives, the parameter is set (to MSB$\times$128) only if the prior CC was also an MSB for the same parameter, or if there has been no prior CC. An MSB must precede any series of one or more LSB. When an LSB arrives, the parameter is then changed to MSB$\times$128+LSB. Modules may inject parameters other than modulation CCs, but should not remove those.

MSB	*Function*
0	Program Save
1	Current Program Save (= 0), Revert (= 1) [Reserved (LSB > 1)]
2–15	[Reserved]
16–31	The first 16 parameters for ID 9
...	
112–127	The first 16 parameters for ID 15

Table 4. Auxiliary Parameters. LSB is the value.

5.4 Program Change and Bank Select

Modules may respond to Program Change messages alone; or include Bank Select (always providing MSB and LSB). Modules responding to Bank Select should do so within some contiguous range $b \in 0...M$ where $b = 128 \times MSB + LSB$. If a Bank Select value is out of range for a given module which supports it, it should also ignore the Program Change. Program and Bank Select may be injected and removed.

Optional Module Select We are considering using the MSB (0) of Bank Select as *Module Select*, used to state *which module IDs* should respond to Program Change, Save, and Revert messages, as shown in Table 5.

5.5 Auxiliary Parameters

The 14-bit CC pair 3/35 provides certain *auxiliary parameters*, as described below. CC 3 (the MSB) sets the parameter number as shown in Table 4, and CC 35 then sets its value. A CC 3 must appear before any series of CC 35: a bare initial CC 35 should be ignored.

CC 3 = 0 is a *Program Save* message. CC 35 (LSB) then gives the value of the program (0...127). The default is 0. This tells a module to save its current state to the specified program in the current bank.

CC 3 = 1 is a directive to *Save to* (value = 0) or *Revert to* (value = 1) the *current* program.

The CC 3 parameters ≥ 16 indicate the first 16 parameters (a...p) for IDs 9...15, and are set only as 7-bit. These parameters should correspond to the first 16 parameters for these IDs in NRPN. CC 35 (LSB) gives their value. To set them in 14-bit, use NRPN.

Reserved parameters and values should be ignored.

We recognize that *Program Save* and *Current Program Save* would represent a break from MIDI tradition in that, excepting System Exclusive directives, there presently exist no MIDI messages which modify the semi-permanent state of the synthesizer, that is, memory that survives power cycling. Thus these directives may be considered still under consideration.

5.6 System and Reserved Parameters

There are sixteen 14-bit and 7-bit CC parameters reserved for system purposes as shown in Figure 2. These include Bank Select, Modulation Wheel, Data Entry, Sustain Pedal, All Sound Off, and so on. All modules may respond to these parameters as appropriate.

The 23 reserved CC parameters should be ignored.

Value	*Function*
0	All Modules on Channel (the default)
1–15	IDs 1 through 15 respectively
16–127	[Reserved]

Table 5. Optional Module Select Values

5.7 Clock, RPN, Sysex, and MIDI Modes

Modules may remove and inject clock signals to perform clock division or multiplication. RPN may be responded to by all modules as appropriate. We note that RPN MPE Configuration parameter may be of particular use to a top-level distribution module. Modules may also respond to System Exclusive messages.

Some modules or voices play notes simultaneously (such as a drumkits), and others play monophonically. The behavior in question depends on the choice of modules for a voice. For this reason, MIDI Modes do not make sense in this context. Modules may be thought of as being in Modes 3 or 4: but at any rate, they should not respond to Mode change CC messages, which are reserved.

6. EXAMPLES

6.1 Example: Trivial Monophonic Oscillator

Consider a square wave oscillator with pulse width controlled by CC. It has sound out and gate out. It cannot be tuned and does not respond to pitch bend. This module would have one MIDI IN and one pure MIDI THRU, with no injection. The module would use OMNI as its channel and could be hard-coded to ID 1, but settable ID would be better. It could respond to Note On/Off and have a CC set to PW/Wave. A better oscillator would also respond to NRPN for PW/Wave.

6.2 Example: Injecting Monophonic LFO

Consider a simple LFO that responds to rate, depth, and type, and that can inject MIDI to a target. It responds to aftertouch to increase the depth, in addition to a CC. It also responds to the Mod Wheel to add to the rate. The module would have one MIDI IN, one pure MIDI THRU, and one MIDI THRU with injection.

Via a switch, we would stipulate which modulation parameter was output, and whether the module ignored (passed) or removed modulation data sent to it by its parent. Via a 3-DIP switch we can set the module ID. It would use OMNI for its channel.

This module would respond to note on, aftertouch, mod wheel, and several mappable CCs for rate, depth, and type depending on the ID, as well as NRPN parameters for the same. It would also respond to (and remove) Modulation CCs for rate, depth, and type.

6.3 Example: Polyphonic Wavetable Oscillator

Finally consider a 4-voice wavetable oscillator, with four independent oscillators on four different voices. The oscillator can respond to pitch, velocity, bend, and portamento. We can set the detune and degree of aliasing in the

wavetable. The oscillator has patches and so responds to Program Change but not banks.

The module would have one MIDI IN and THRU that apply to all channels as well as, at its discretion, four MIDI IN/THRU socket pairs to work in conjunction with monophonic voice chains. Each oscillator could have its own ID, but normally they would use the same ID. The module would respond to note on and off, mappable CCs for Osc Tuning, PW/Wave, Wavetable, and aliasing, as well as All Sound Off, All Notes Off, and NRPN. It would also respond to Program Change.

7. DISCUSSION AND FUTURE WORK

This proposal is version 0.5. Beyond the MIDI modules already available commercially, we have implemented preliminary versions of this proposal in hardware, including CC/NRPN response and message injection or removal. We expect to need to make changes to the proposal, and publish several revisions.

We recognize that one issue with this proposal is that, in order to deal with the dual-namespace issue, it must use much of the CC parameter space. There are only 128 CCs to go around. We have tried to be flexible and to allow NRPN and Auxiliary Parameters, but may need to add additional tricks in the future. The proposal also does not define a full graph structure. We have attempted to allow modules to modulate one another while remaining true to MIDI 1.0 standards, and permit (but avoid) MIDI Merge; but we recognize that there may be rare complex configurations which are awkward to implement in our scheme.

This proposal was developed for and by the AE Modular community but it is not in any way limited to AE Modular. Nothing in the spec precludes porting to other platforms, like Eurorack. We very much would like to see it used on those platforms, or inspiring similar designs.

Acknowledgments

Our thanks to Beppe Sordi, Luca Ludovico, and the AE Modular community for their help, criticism, and feedback, and to LIM at U. Milan for its support.

8. REFERENCES

[1] *The Complete MIDI 1.0 Detailed Specification* (Version 96.1, Third Edition), MIDI Manufacturers Association, Los Angeles, 1996.

[2] H. Bode, "A new tool for the exploration of unknown electronic music instrument performances," *Journal of the Audio Engineering Society*, vol. 9, no. 4, pp. 264–266, 1961.

[3] Buchla Electronic Musical Instruments, "User's guide for the 200e Electric Music Box (v1.4)," buchla.com/guides/200e_Users_Guide_v1.4.pdf, 2017, as of 1/1/2024.

[4] "Eurorack modular system preset protocol," docs.google.com/document/d/1YhPvAI6oliwLYSHhDholdAUym-TfBDGlxST-zDCP-qw, 2016, as of 1/1/2024.

[5] "CVM-8 firmware v1.1 user manual," www.expert-sleepers.co.uk/downloads/manuals/cvm-8_user_manual_1.1.pdf, 2022, as of 1/1/2024.

[6] "Monome Teletype," monome.org/docs/teletype/, as of 1/1/2024.

[7] "ART: Chords, melodies, harmony, and polyphony in Eurorack," tiptopaudio.com/art/, as of 1/1/2024.

[8] "Morningstar Engineering," www.morningstar.io, as of 1/1/2024.

[9] J. Tuffen, "MIDI: a new signal type for AE," wonkystuff.net/midi-a-new-signal-type-for-ae/, 2023, as of 1/1/2024.

[10] "Tangible Waves (AE Modular)," www.tangiblewaves.com/, as of 1/1/2024.

[11] *MIDI Polyphonic Expression (Version 1.0)*, MIDI Manufacturers Association, Los Angeles, 2018.

[12] S. Jungleib, *The Complete SCI MIDI*, Sequential Circuits, Inc., 1983.