Gareth Loy
Computer Audio Research Laboratory
Center for Music Experiment
University of California, San Diego
La Jolla, California 92093 USA

Musicians Make a Standard: The MIDI Phenomenon

Introduction

MIDI (Musical Instrument Digital Interface) is a specification of a communications scheme for digital music devices. Like all products of human cunning, it has its good aspects, and bad ones. Its advent is more notable for the effect it has had on the music community than for its prowess as a network. Whether MIDI gets good or bad reviews depends to some extent on whether it represents a step up or down in expressive potential from the system the reviewer is currently using. It makes some happy that a networking standard exists for music instruments; it makes others frustrated that it has so many limitations. The limitations include limited bandwidth between devices, limited frequency and time resolution, limited access to synthesizer parameters for such things as timbre modification during synthesis, and lack of bidirectionality in communications. The conception of music embedded in the standard seems archaic and inflexible, and favors piano-keyboard type synthesizers. However, MIDI has flourished and is now the *de facto* industry standard. In spite of its limitations, it is quite serviceable for a variety of tasks. Its usefulness comes in no small part from its being a standard, whatever its limitations. Its success has sparked interest in the development of other standards for domains such as music databases, editing, and for extensions or replacement of the MIDI standard itself.

This article is something of a survey, tutorial, and review, all mixed together. Under discussion are the following topics:

The reference model for MIDI
The MIDI 1.0 specification
The control paradigm MIDI implements
The implications of this paradigm for performance capture and synthesizer control
Conclusions regarding its usefulness for the variety of different tasks in which it could find application

Beyond this, we will look at the forces that brought MIDI into being, and consider what the future holds, now that we know that MIDI will be a part of it.

The basic message is that while MIDI has been characterized as rock-n-roll's answer to computer music, this condescension is not warranted. If the number of articles published on the subject of music networking is any indication (Bischoff, Gold, and Horton 1978), the subject has been almost completely ignored. This will change now that there is a common technological base from which to work. For instance, networking is very important to the study of live performance, since the latter can be viewed in terms of the former.

In spite of its limitations, MIDI provides tools that can be helpful to the study of performance practice, computer-assisted performance, and improvisational composition. Simple as it is, it has intrinsic worth for practicing musicians and composers. Grasping the lessons that the MIDI phenomenon has to teach will be revealing no matter what its direct usefulness. It will probably lead more people to more new insights about music performed live with computers than anything since the GROOVE system (Mathews and Moore 1970). To this end, my hope is that the serious study of MIDI will lead to its own betterment.

Motivations for MIDI

MIDI was developed by several commercial synthesizer manufacturers over the last few years. The original motivation was to allow commercial synthesizers to be connected together so that they

Computer Music Journal, Vol. 9, No. 4, Winter 1985,

might share control information, such as the gestures of a musician. Other benefits sought included hardware extensibility, protection from obsolescence, and interfacing to digital computers. The latter can provide for multitrack recording/playback, sequence editing and composition, score display, and music printing.

It is important at the outset to say that MIDI is designed as an *event-based* network, not a *sample-based* one. What MIDI communicates is not sampled waveforms, but indications about pressed keys and switches, turned knobs and pedals—in other words—human gestural control information.

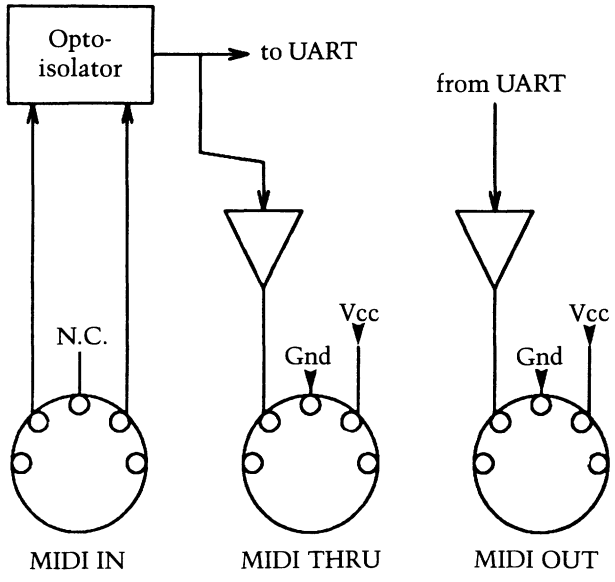
MIDI Specification

The first order of business is to present the MIDI 1.0 specification, which is given here in an abridged form, suitable for the issues under consideration.[1] While I have tried to keep back my observations until the concluding section, it seems that some blows must be struck while the iron is hot; so where I have been unable to restrain myself from critical observation, I have enclosed these comments in braces {}.

MIDI was loosely adapted from the serial data transmission technology developed for computer terminals. The basic idea involves a two-layer specification: a physical interconnection scheme, and a code to communicate information across the channel so created. Obvious requirements for the physical layer are that it must be rugged, capable of driving signals over medium distances without loss, resistant to electrical and magnetic interference, noninterfering so as not to pollute nearby analog signals, and capable of the requisite transmission bandwidth. The code layer must be as concise as possible while still permitting all the forms of expression required by the communicating devices. It must be extensible, fault tolerant, and yet efficient. Both layers are also constrained to be extremely

1. Readers interested in obtaining a copy of the complete specification should contact the International MIDI Association, 11857 Hartsook St., North Hollywood, California 91607, telephone (818) 505-8964.

Fig. 1. Block diagram of MIDI transmitter and receiver circuitry.

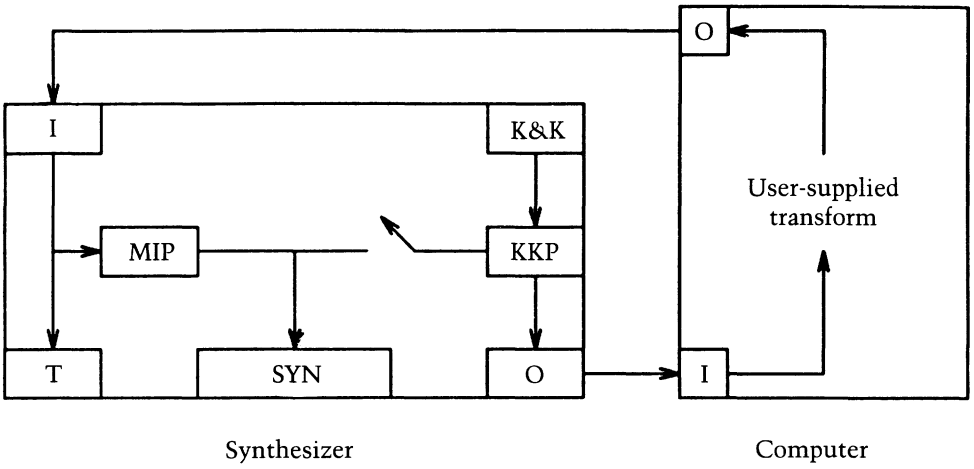


cost-effective, meeting the demands of high volume manufacturing. While we will find that MIDI satisfies these criteria, later we will discuss some other criteria that were overlooked.

The Physical Specification

The physical medium is a simple point-to-point opto-isolated 5 ma current loop utilizing a unique 180 degree 5-pin DIN connector (Fig. 1). Only three of the pins are used. Cannon XLR connectors were originally specified as an alternative suitable connector, but this was subsequently dropped. The cable is made of a shielded twisted pair, the shield being grounded only at the source end. Each twisted pair is a separate run that implements a one-direction transmission line. MIDI devices are to have a MIDI input, and output jack. In addition, devices can have a MIDI THRU (through) jack, which passes a buffered electrical copy of the input signal. (Note: MIDI input is not connected to MIDI output, but to THRU.) Information is transmitted as asynchronous serial data at an aggregate data rate of 31.25 Kbaud (31250 bits per second). {No one seems to know for sure why this particular figure was

Fig. 2. Functional components of a conventional MIDI synthesizer. Shown here connected to a computer, the standard MIDI synthesizer functions include I, O, *and* T *(MIDI IN, OUT, and THRU, respectively);* MIP *(MIDI input processing);* K&K *(keys and knobs);* KKP *(key and knob processing); and* SYN *(the synthesis engine).*



chosen, but it is noted that it equals a 1-MHz clock divided by 32.} Serial MIDI data is transmitted as ten-bit code bytes, consisting of a start bit, eight data bits, and a stop bit. This results in a 320-μs-byte transmission time. The outer two framing bits are added by the transmitter for synchronization and are stripped off by the receiver, leaving a conventional eight-bit byte. Typically, a UART (Universal Asynchronous Receiver/Transmitter) or ACIA (Asynchronous Communications Interface Adapter) chip can be used to convert from parallel to serial data formats. Some microprocessor chips contain on-chip serial input/output (I/O) ports and timers suitable for this purpose.

The "reference model" for MIDI only defines the network. However, several aspects of the synthesizer are implied by the specification. The specification says only that a synthesizer shall have a MIDI receiver, a MIDI transmitter, and optionally, a MIDI THRU port. An electrical copy of the signal sent to the MIDI receiver is passed to MIDI THRU. The synthesizer is presumed to have on-board knobs and switches, and some sort of synthesis engine. Ordinarily, the knobs and switches connect both to the synthesis engine and to the MIDI transmitter (Fig. 2). {This standard layout—emulated by most manufacturers—provides no way to decouple the synthesizer's on-board controls from its synthesis engine. This precludes the insertion of an external computer between the controls and the synthesis engine of a single instrument. Such synthesizers are conceptually similar to half-duplex computer terminals. There is a command in the MIDI specification to break the internal connection between controls and synthesis (described later), but it seems to be rarely implemented.}

Some interconnection schemes for multiple synthesizers are as follows:

Unidirectional—master talks to slave.
Bidirectional—two masters drive each other as slaves.
Ring—an extension of bidirectional connection to three or more devices.
Daisy chain—one master drives several slaves using MIDI THRU.
Star—one master has several unidirectional or bidirectional links.

These interconnection schemes are shown in Fig. 3–6.

In the general case, we see that the IN/OUT/THRU connection scheme forms a triple that joins with other triples to form a binary tree. This is illustrated in Fig. 7. The left branches receive MIDI input only from the root of the tree via MIDI THRU, plus they respond to their own controls. Right branches inherit MIDI input only from their immediate parent, but pass it along to all

Fig. 3. Multiple master/slave configuration. Each box is a MIDI synthesizer expressed as a triple: T, I, and O are MIDI THRU, IN, and OUT, respectively. Synthesizers are labeled to the left with a capital letter; the expression to the right indicates the source of control information for each synthesizer.

Fig. 4. Dual master/slave configuration.

Fig. 5. Ring configuration.

Fig. 6. Star configuration. M is a master controller.

Fig. 3

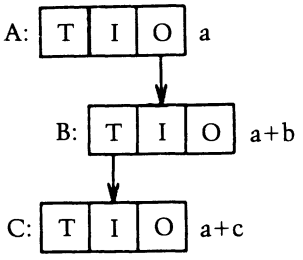


Fig. 4

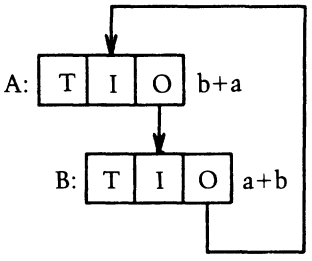


Fig. 5

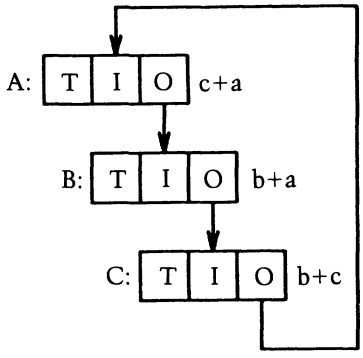
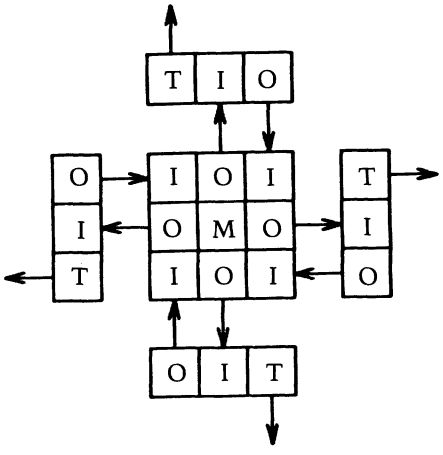


Fig. 6



left-branch children, while passing the result of their own control input only to their right-branch children.

{It is important to emphasize that a single MIDI cable is a unidirectional single-talker/single-listener network. Using a MIDI THRU connection, it can become a multilistener network, but there is always just one fixed transmitter for any subtree. What is more, there is nothing in the specification that even hints at two-way communication. Thus, there are no provisions in the specification for methods of interrogation and response of networked synthesizers. The significance of this limitation is elaborated later.}

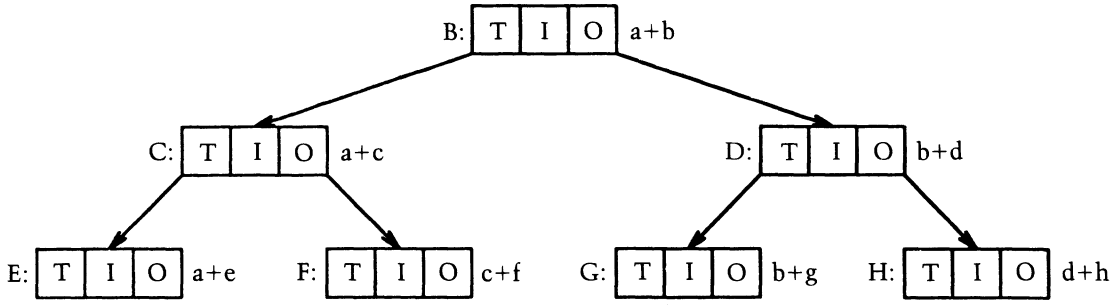
The Code Specification

The other half of the specification details the nature of the information that is communicated over the physical medium. The code specification consists of three elements, *modes*, *channels*, and *commands*.

MIDI Modes and Channels

There are three modes and sixteen channels. The channels provide for multisynthesizer control with a single MIDI network, while the modes establish the relationship between the channels and the voice-assignment method within a synthesizer. The term "channel" is confusing to those who are familiar with the term as used in the recording industry. {Many terms used in the MIDI specification are only abstractly related to their more conventional meaning, as we see shortly.} Here is an explanation of MIDI's notion of channels. Many MIDI commands, collectively known as *channel commands*, have a field for a channel number. Synthesizers can be configured to receive or ignore channel commands depending on the channel number. This can

Fig. 7. The general case of MIDI interconnection forms a binary tree. The left branches receive MIDI input only from the root of the tree via MIDI THRU, and they respond to their own controls. Right branches inherit MIDI input only from their immediate parent, but pass it along to all left-branch children, while passing the result of their own control input only to their right-branch children. Synthesizers are labeled with capitals to the left of each box; the sum of their input and local controls is expressed in lowercase to the right of each box. An implicit synthesizer A (not shown) is connected to the input of synthesizer B. The polynomial at the bottom of the figure shows the number of synthesizers playing each part.



$3(a+b)+2(c+d)+e+f+g+h$

be used to restrict which synthesizer(s) will respond to a particular channel command.

The modes are called *omni*, *poly*, and *mono*. *Omni* mode causes a MIDI unit to accept commands on all channels. The synthesizer transmits on only one channel. In *poly* mode, each MIDI unit only receives and transmits commands on one channel. Voices are assigned polyphonically; that is, sequential *note on* commands generate chords. *Mono* mode is not what it sounds like: it is supposed to mean "one voice per channel." A synthesizer capable of *mono* mode operation will be assigned a range of channels (perhaps only one) to which it is to respond. Commands for each channel will control a single voice. *Mono* mode is used to provide for certain portamento effects not obtainable with *poly* mode. Here subsequent *note on* commands could cause a glissando to the new note instead of a chord. Also, for polytimbral synthesizers—that is, ones capable of generating more than one timbre at a time—when *mono* mode is used, the different timbres of the single instrument can be assigned to particular channels.

The modes are grouped into four states. {The MIDI specification refers to these states as "modes" as well, a confusion I have sought to avoid.}

State 1 (*omni on* and *poly*). All commands regardless of channel are recognized by the receiver and assigned to voices polyphonically.

State 2 (*omni on* and *mono*). All commands regardless of channel are recognized by the receiver and assigned to one voice. Only one voice sounds.

State 3 (*omni off* and *poly*). Only channel commands matching that of the receiver are recognized and assigned to voices polyphonically.

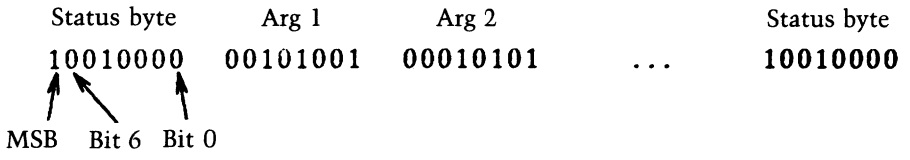
State 4 (*omni off* and *mono*). Channel commands that match a range of preselected channels are recognized and are assigned one channel to one voice.

{Dynamic voice allocation in *poly* mode can be problematic. If all voices are in use in a synthesizer and another *note on* command arrives, the synthesizer must decide how to cope with the new demand. While there are numerous solutions, the one most often chosen by manufacturers is to have the synthesizer steal the voice of the oldest sounding note on the presumption that the attention of the listener has been shifted away from it by subsequent notes. This method fails badly when the note being stolen is, for example, a pedal point, in which case its absence becomes quite glaring. This is an example of a problem that requires musicological awareness to solve.}

Where there are modes and channels, there are, or should be, means to change them. {Alas, while there are *mode select* commands, there are no *channel select* commands.} Mode select commands target selected channels, and the synthesizers on those channels are to switch to that mode if they can. {But if they cannot, their failure to do so will never be reported because MIDI is a unidirectional network.}

MIDI provides no method to change channel assignments. They must be changed by physically

Fig. 8. MIDI byte stream sequence.



manipulating the synthesizer. Some synthesizers only transmit on MIDI channel 1 but can receive on any channel. Channel 1 is supposed to be the power-up default, but many manufacturers simply remember the last set channel across power cycles. This means there is no way for a MIDI network to be configured under program control; it must be done manually.

MIDI Command Types

A MIDI command is a sequence of one or more bytes of data. There are five categories of MIDI commands: *channel*, *system common*, *system realtime*, *system exclusive*, and *reset*. The *system realtime* category is highest priority, followed by *system exclusive*, with the rest grouped below them. *System exclusive* commands cannot be interrupted by any lower priority command. *Realtime* commands can interrupt any multibyte commands. The salient characteristics of the command types are as follows.

Channel commands communicate event data, such as *note on/off* and status of device controllers.

System common commands deal with sequence selection and location within the sequence. This is mostly for MIDI sequencers, which are devices that can store and regenerate MIDI commands.

System realtime commands are for synchronizing a network of MIDI sequencers to a common clock.

Reset terminates any activity in progress and reinitializes to the power-on condition.

System exclusive is for device-specific data transfer, for sending patches, parameters, etc. The format of *system exclusive* only designates how it starts and where it ends; the content of the transfer is unspecified and is presumed to refer only to one vendor's equipment. Part of the preamble for *system exclusive* is a byte containing a unique manufacturer's identification number. Synthesizers matching that number know to respond to the command, others ignore it. {*System exclusive* is the great escape hatch of MIDI. There are strong tendencies to solve problems by dumping solutions under this category of command. This both subverts what little standardization MIDI provides and helps guarantee that it will never be improved upon.}

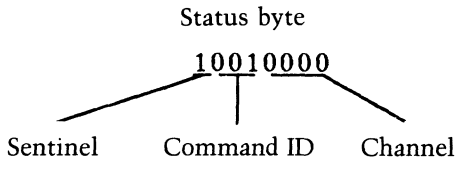
Format of Commands and Data

Figure 8 illustrates the bitwise sequence and layout of a MIDI command, after the start and stop bits have been stripped off. The most significant bit (MSB) is a *sentinel* bit, which signals the beginning of a new command if it is set. If set, the rest of the byte is decoded as a status byte. Bits ⟨6:4⟩ contain a partial command type identification (ID). If they indicate a *channel* command, then bits ⟨3:0⟩ are the channel number of the command, otherwise bits ⟨6:0⟩ are taken as the complete command type identification (Fig. 9).

MIDI specifies the number of trailing data bytes for each command type. Some commands take no data (such as *reset*), some one byte, some two, some any number. A new command is defined to commence whenever the sentinel bit of any byte is set. This restricts data to occupy the low-order 7 bits of each data byte. Table 1 provides a tabulation of the MIDI codes, their meanings, and the number of data bytes and their meanings. The Appendix provides additional descriptions of the commands.

{Note that the *control change* command in Table 2 includes mode changing, *all notes off*, and local/remote controls as well as device controls. It strikes me that this is an unhappy mixture of meta-

Fig. 9. Channel command layout.



phors. There should have been a separate command dealing with global status and control.}

The specification says that when a subsequent command is identical to the preceding one (same command ID, same channel), the status byte need not be sent. This means that after the first command, only data bytes need be sent until a command with a different status byte must be sent. This can significantly reduce the amount of data transferred, but it depends upon the statistical properties of the command stream how much savings is realized.

Interfacing MIDI to Computers

Insofar as MIDI implements a very similar physical discipline to standard computer terminal interfaces, one approach to interfacing MIDI devices to computers is to modify a standard computer terminal line. The adjustment includes modifying the baud rate to 31.25 Kbaud, and installing an opto-isolator. However, the host computer would be dealing with a large volume of time-critical, continuous programmed I/O to support MIDI. This would place a major load on most microprocessors, reducing their ability to simultaneously address other tasks. It is better to offload the task to a peripheral processor designed to interface to MIDI on one side and to a computer on the other.

I have worked with two such devices, one made by Sound Composition Systems (SCS), the MIDI Performer, the other a Roland MPU-401 MIDI Processing Unit.[2] Both are small boxes containing microprocessor systems with I/O ports for MIDI and host computer communications. The host computer sets the unit to perform an operation such as record or playback, and the unit then interrupts the host only when MIDI commands or data are available or needed. This frees the host to perform higher level support functions such as compositional algorithms, analysis, display, and so on. The SCS device communicates with the host via a standard RS-232 computer terminal interface while the Roland unit interfaces directly to the bus of the host computer. By using a standard computer terminal interface, the SCS device can be hooked up to just about any computer using whatever driver software exists in the host for talking to terminals. Roland provides bus interfaces for some personal computers (including the IBM PC and Apple II). The interface itself is extremely trivial, and interfaces to other busses are easy to build, but it is not so easily integrated as the SCS, since it also requires that a device driver for the unit be written for the host computer.

The Roland unit, by going straight to the host computer's bus, does not have bandwidth problems as does the SCS unit. The SCS interface problem is a result of the difference between the 31.25 Kbaud MIDI rate and its 9600 baud host computer communication rate. Data overrun from a MIDI source to the SCS unit is quite likely when recording MIDI data, since the SCS unit can only pass along the incoming MIDI data to the host at about a third of the speed of MIDI. While the SCS unit can buffer up to 8 Kbytes of MIDI commands, it will always remain a statistical question as to whether this will be sufficient for any particular MIDI command stream. For this reason, one is usually driven to use the methods SCS provides for filtering out continuous controller information, such as pitch bend and modulation, passing only the lower bandwidth event-oriented commands such as *key on* and *key off*. However, the necessity of doing this limits the unit's generality. The SCS unit could presumably run its host-computer link faster than 9600 baud (e.g., at 38.4 Kbaud). However, the software provided on most computers to interface to standard computer terminals is not designed to handle sustained high transfer rates. Also, UARTs that support

2. Note: Sound Composition Systems (Pasadena, California) has apparently gone out of business. A new company called Hinton Instruments (Oxford, England) makes a serial-to-MIDI interface called MIDIC (see *Computer Music Journal* 9[2]:71). The Hinton box may not suffer from the same throughput problems as the SCS box did because the Hinton box will run at 38.4 Kbaud.

Table 1: Summary of MIDI codes

Status	*Arg 1*	*Arg 2*	*Mnemonic*
		CHANNEL COMMANDS	
80	Key	Velocity	Key off
90	Key	Velocity	Key on
A0	Key	Pressure	Polyphonic key pressure (*after-touch*)
B0	Index	Value	Control change
C0	Index	(None)	Program change
D0	Pressure	(None)	Pressure (*after-touch*)
E0	LSB	MSB	Pitch wheel change
		SYSTEM EXCLUSIVE COMMAND	
F0	Mfg. ID	. . .	System exclusive command
		SYSTEM COMMON COMMANDS	
F2	LSB	MSB	Program position select
F3	Index	(None)	Program select
F6	(None)	(None)	Tune request
F7	(None)	(None)	End of system exclusive
		REALTIME COMMANDS	
F8	(None)	(None)	Timing clock
F9	(None)	(None)	Undefined
FA	(None)	(None)	Start
FB	(None)	(None)	Continue
FC	(None)	(None)	Stop
FD	(None)	(None)	Undefined
FE	(None)	(None)	Active sensing
FF	(None)	(None)	System reset

Note: All values are in hexadecimal notation. See the text for a further description.

rates beyond 19.2 Kbaud are still somewhat rare. One must trade off the simplicity of the SCS hardware interface against these considerations.

Storing MIDI command streams on disk in a host computer necessitates time-stamping the MIDI commands. Both the SCS and the Roland prepend a time stamp to all recorded MIDI commands, and expect a time-tag to precede all MIDI commands played back. The SCS unit provides a 16-bit resolution time-tag, while the Roland provides 8 bits. The tick time is adjustable, with usable values in the range of 1–5 msec. The SCS unit does not provide a clock-continuation command, presumably figuring that 16 bits of clock resolution ought to provide enough distance between any two MIDI commands for their standard clock time. Roland's 8-bit counter resolution only covers about 1.2 sec at their default clock rate, requiring that the number of clock overflows occurring between commands be kept. Roland uses the MIDI *timing clock* command to indicate clock overflows. While both methods seem adequate, I favor Roland's. For a performance of average

Table 2: Control change command indexes

Status	*Index*	*Argument*	*Meaning*
D0	0 ⇔ 1F	Value	MSB of continuous controllers
D0	20 ⇔ 3F	Value	LSB of continuous controllers
D0	40 ⇔ 5F	Value	Switches
D0	60 ⇔ 79	Value	Undefined
D0	7A	(Value)	Local control (7f = on, 0 = off)
D0	7B	(0)	All notes off
D0	7C	(0)	Omni off (All notes off)
D0	7D	(0)	Omni on (All notes off)
D0	7E	(Channels)	Mono on (Poly off) (All notes off)
D0	7F	(None)	Poly on (Mono off) (All notes off)

Note: All values are in hexadecimal.

musical density, it is likely that the Roland format will be more economical in bytes transmitted. Also, there is no limit to the size of the interval between two MIDI commands in the Roland format.

A difficulty with both methods of time tagging is that the time-tag can legally have the MSB set, leading to uncertainty when trying to parse corrupted data. This is no problem for well-formed command streams. However, if a stretch of bad data is encountered, one can't simply look for the next byte with the MSB set in order to resynchronize on a status byte, as this might be just a time-tag. One must also look at the byte following to make sure its MSB is off.

The protocol through which the MPU-401 communicates to the host is rather convoluted. Instead of communicating state information to the host as to its place within the commands being processed, the MPU-401 requires that the host essentially duplicate the internal state of the MPU in order to know what is going on. As a result, a device driver that avails itself of all of its diverse capabilities will be relatively complicated.

Packaged computer/synthesizer/software systems are beginning to appear using these interfaces and others like them. Other vendors include Passport Designs, Musicdata, Hybrid Arts, and Jim Miller, an independent. Without a computer, about all one can do with MIDI is double voices. In order to get any compositional or editorial power, one must have a general-purpose computer (or a MIDI "sequencer," which is usually a general-purpose computer with a custom user interface and MIDI I/O). Interfaces for computers usually target home computers, such as the IBM-PC, Apple II, Apple Macintosh, Commodore 64, etc. Some microcomputers from Yamaha and Atari are now being manufactured with a MIDI interface as a standard peripheral port. In other cases, a synthesizer is integrated directly into the microcomputer, such as the Yamaha CX5M. [Reviewed in *Computer Music Journal* 9(3)—Ed.] In addition, a multitude of homebrew interfaces are being constructed for everything from mainframes to lap computers. One company, J. L. Cooper, markets MIDI interfaces and converters, including devices that use MIDI to control theater lighting.

Packaged software for these systems provides for basic record/playback functions plus simple editing. Most of the software adopts the metaphor of the multitrack tape recorder to structure the user interface. Thus, editing takes the form of "fast forward/rewind" and "punch in." Event-time correction is usually provided to "discretize" notes to the nearest selected integer ratio of the beat. Some packages provide means to display MIDI data as common practice music notation. Results vary widely with the adequacy of the graphics, user interface, and the grip of the programmer on basic musical issues.

Most packaged systems for MIDI, like MIDI itself, are aimed at the working pop musician. As such, most are closed, proprietary systems. Few, if

any, provide a means whereby someone could adapt or extend a system to their own ends. Of course, absolutely none of them were written with the idea of providing a development environment for computer music research.

Misconceptions About MIDI

The foregoing is a relatively complete summary of the MIDI specification. However, there are still things that neither the specification nor my explanations have yet made clear. I will attempt to shed light on these by discussing some common misconceptions. Regardless of what it is, MIDI serves no one if it is perceived for what it is not.

Misconception 1: **MIDI is a bus.** It is not. A bus implies bidirectional communications and the possibility of more than one bus master. MIDI is a unidirectional talker-listener network. The term "network" is even a little broad for MIDI. The MIDI specification does not preclude devices from being connected for interrogate/response communication, but this form of communication would require that the specification be extended to develop a vocabulary of what they could say. Such a capability could be used, for instance, by a master to configure a MIDI network automatically, by invoking responses that would return manufacturer's ID, channel number, and reception mode. Based on this information, the master could then emit change-mode commands, and (if they existed) change-channel commands. This is just for starters; the compositional and performance possibilities for true networking are endless (Bischoff, Gold, and Horton 1978).

Misconception 2: **MIDI does not have sufficient bandwidth to capture human performance.** If we restrict ourselves to keyboard instruments, the experience of musicians I know who have used it is that it does have acceptable bandwidth. Its very success is a kind of proof of this, but let's take a closer look. As we have seen, a byte can be transmitted every 320 μsec over MIDI, and it typically takes 3 bytes per command, which gives us 1 msec per command, worst case. Let us take for example a keyboardist striking ten keys simultaneously. This produces ten *note on* commands, which serialized would be smeared over 10 msec. (This is assuming the unlikely event that all ten fingers managed to strike the keyboard simultaneously. This also does not account for the MIDI command-continuation feature, which would reduce the transmission time to 7 msec.) When compared to an average attack time for percussive instruments of 10 msec, we see that the smearing is not liable to be heard as multiple attacks, even when percussive timbres are being synthesized.

Another way to view this is to realize that sound travels about ⅓ meter per millisecond. A delay of 10 msec represents a distance between musical sound sources of 3 m. Amplified musicians typically do not complain about being this distance from their loudspeakers. Also, this can be compared to the size of a symphony orchestra, which can successfully fuse a musical percept while flung out over more than 30 m.

These are merely analogies, however, and this is not to say that millisecond-level serialization delays have no effect, only that they have no effect for the case of a keyboardist using MIDI. Recently, my colleague F. R. Moore did a simple experiment to observe the effect of such delays on timbre. He generated a waveform consisting of two single periods of a 1000-Hz sine wave which were separated by a 1-msec silent interval. He then synthesized another sine wave pair separated by 2 msec, continuing this until he had pairs separated by all intervals between 1 and 10 msec. The pairs were separated by 1 sec of silence each. What was heard when this was played was a sequence of clicks, where the pitches of the successive clicks described a subharmonic series. He concluded that while serializing commands does not have the effect of producing multiple attacks, it does have an effect on timbre, similar to comb filtering. Issuing MIDI commands that cause two instances of the same exact waveform to be started 1 msec apart is equivalent to applying a one-shot delay of 1 msec to the first one.

Because of the crude nature of time control, MIDI is utterly inadequate for phase-level control of waveforms. This means synthesis of a single-fused timbre cannot be split reliably across MIDI synthesizers. This would also rule out control of stereo

or dichotic sound imaging via MIDI. Even though the effect of the serialization delays can occasionally be seen as a plus, as when chorusing effects are desired, however, we must not confuse an inherent artifact of a system with a feature of the system. The important question is not "does it matter?" so such as "when does it matter?"

In addition to event data, MIDI can be used to transmit continuously sampled data. The 7-bit sampling rate is 3125 Hz, while the 14-bit rate is 1562 Hz. While this is very slow for acoustic pressure functions, it is not too bad as a control rate for limited applications. For comparison, we can recall the GROOVE system at Bell Laboratories (Mathews and Moore 1970), where it was experimentally determined that functions of time representing general human performance gestures could usually be adequately represented at a sampling rate of about 200 Hz per function. Assuming this is correct, it shows that, for the worst case, MIDI is capable of transmitting nearly eight 14-bit functions at the rate of 200 samples per second. This means, for instance, it could presumably handle continuous pitch change information for all strings of a violin or guitar. However, this leads us to our next assumption.

Misconception 3: **Existing commercial synthesizers run at full MIDI bandwidth.** Some do, maybe, but not the ones I looked at. I discovered this while researching the question of command smearing for Misconception 2. I attempted to hear the result of ten simultaneous attacks by transmitting first one, then two, and so on up to ten *note on* commands at maximum MIDI bandwidth to a well-known MIDI synthesizer from a host computer. The results were surprising, to put it mildly. The synthesizer required an average of about 2.5 times as long to turn on a voice as to receive a MIDI command. In the worst case this was 20 msec to turn on six voices. Mysteriously, it ran faster turning on 10 voices (taking about 17 msec). I then tried another well-known synthesizer with mixed, but generally poor, results. This experiment was limited and informal, but it points up that performance is not dictated by the MIDI specification.

Misconception 4: **The data rate requirement for performance gesture capture is the same requirement as that for synthesizer control.** MIDI builds this assumption into the specification. In fact the data rate requirement for synthesizer control is usually far greater than for gesture capture. The one instance where the rates are equal is where a straight performance recording is made from a gesture input device to a host computer which then simply regurgitates the performance. However, this is the most trivial use of MIDI. More likely uses include overdubbing and computer-generated scores, both of which would typically involve much greater bandwidth than that generated by a single performer. Indeed, a critical aspect of any such system would be that it does not limit the output to the capabilities of performers!

Another strong temptation will be to use MIDI *system exclusive* commands to modify synthesizer parameters during performance execution to simulate more interesting control functions than are available in the synthesizer's hardware. Thus the host-computer-to-synthesizer direction will tend to be denser than the performance-input-to-host-computer direction. The density will grow with increasingly sophisticated use of MIDI. That MIDI does not address this obvious problem stems from MIDI's apparent origin as a gesture capture mechanism, for which the prevailing low data rate seemed acceptable.

This problem will probably not deter people from trying to squeeze blood from this turnip by using MIDI to control ever larger arrays of more and more capable digital music devices. The results obtained are likely to be sufficient for small networks and initial efforts, but as the idea of music instrument networking takes hold, I expect to see serious attempts to address the bandwidth limitations.

Another concern that falls into this category is the quantity of synthesis resources a single performer is capable of manipulating in real time. In the case of keyboards, the temptation is to apply the "ten finger" rule: "nobody can play more than 10 fingers at once," as a way of justifying low numbers of polyphonic voices. However, standard Romantic keyboard literature (e.g., Chopin) shows that this rule does not hold as soon as the sustain pedal is depressed. On pianos, the sustain pedal lifts the dampers on the strings, causing all notes initiated to be sustained while the pedal is down. It is not clear how many voices are required by such music, but a good guess is no less than 88, the number

of keys on a standard piano. Also, consider conducting: a wave of the hand can result in a mass of instruments playing. When musicians are given computer control over synthesizers, the natural course of things is to require ever more control over ever more resources.

What, Then, Is MIDI Good for?

MIDI is certainly good for what it was designed to do and to be. It was designed by manufacturers to extend the means of control over their synthesizers in a device-independent and vendor-independent way. It is standard, simple, inexpensive, and effective at what it does. These positive achievements cannot be denegrated. Beyond the obvious bugs in the specification, most of the difficulty with MIDI comes not from MIDI but from the attempt by others to make it do that for which it was not designed.

Like most things, MIDI was not designed with the full significance of what it could be in mind. MIDI has opened a Pandora's box of possibilities for music instrument networking that were either not forseen by its developers, or were ignored. The problem is that none of these dreams will now fit back into the box. It seems that we can either ignore the issue altogether, make do with MIDI, or try to make a more adequate vehicle for our networking aspirations.

What Are the Chances that MIDI Can Be Improved?

All who talk of change to the MIDI specification must first examine the forces that brought it into being. There are two ways that standards come into the world. A formal standard is wrought by a committee working under the auspices of an organization such as the American National Standards Institute (ANSI). On the other hand, an informal standard is worked out by a group of practitioners in need of a common approach. MIDI is clearly in the latter category. The MIDI specification came from an informal group of manufacturers working together. Today it is the province of a group known as the Manufacturers MIDI Association (MMA), whose authority comes from their joint share of a majority of the commercial synthesizer market. Membership is restricted to commercial interests such as manufacturers, software houses, and systems integrators. Another major force in this arena is the Japan MIDI Standards Association, which works closely with the MMA.

The manufacturers have a considerable stake in the commercial success of MIDI and have an understandable unwillingness to fiddle with it. They are unwilling at this time to go beyond ironing out the remaining ambiguities in the standard, and making sure that all vendors' synthesizers will work together under MIDI. Even this little should be considered an heroic accomplishment, considering the distance they have come to get as far as they have. It was by no means a foregone conclusion that the "iron curtains" that existed between manufacturers could be broached by MIDI.

The International MIDI Association (IMA) is the "users group" for MIDI. Started in 1983, this group describes itself as

> a non-profit organization dedicated to the evolution, integrity and continuity of the Musical Instrument Digital Interface. IMA maintains a non-competitive organization interested in the accurate dissemination of information concerning all aspects of MIDI-related instruments and products. The International MIDI Association believes in the privacy of its membership and the protection of proprietary information (IMA 1983).

They publish a newsletter to members called the *IMA Bulletin*, which contains product announcements, reviews, short articles, tutorials, and broadsides.

What the manufacturer's group refers to as "independents" are the most active constituency of the IMA. These are individuals and small businesses who have an interest in utilizing MIDI, either commercially or musically. This is the group that most actively presses for a more adequate networking standard, since it is this group that, by working with MIDI, experiences its limitations most keenly.

Although the IMA has attempted to address issues of standardization and the limitations of MIDI,

they have not received much support from the manufacturer's group, as many expected. For instance, in the spring of 1984 the IMA sponsored a conference called MIDISOFT-84 at the Mark Hopkins Hotel in San Francisco. The population of the conference consisted almost entirely of independents, with a sprinkling of engineers and management from the more enlightened manufacturers. Papers centered on two issues, how to exploit the potential of MIDI, and how to formalize standards. Much rhetoric was spent on lamenting the polarity between manufacturers and independents. This was interspersed with some very insightful comments on the current state of affairs, and on the likely future directions, some of which I am repeating here.

Yet it would be wrong to view the manufacturers as inextricably locked into the current MIDI specification. One must assume that the manufacturers see the limitations of MIDI as a threat to their own well-being in the long run. The worst-case scenario would be if individual manufacturers attempted to undercut the MIDI standard with proprietary networking improvements for their own line of machines. One of the most profound impacts of MIDI on the manufacturers is that it has converted a vertically integrated market into a horizontally integrated one. Before MIDI, each manufacturer produced product lines that were self-contained and often incompatible with the product lines of other manufacturers. Now manufacturers can no longer count so heavily on sales of one item of their product line carrying a package sale. The possibility of both improving MIDI and regaining a vertical market must be sorely tempting. This, however, would be a fatal way to improve MIDI. The fact of the matter is that independents and musicians using MIDI depend for progress on a strong, cohesive manufacturers' MIDI group that can impose change in a uniform way.

What to Improve about MIDI and How

Politics aside, MIDI will certainly change sometime, or be superseded. When that happens, what will we have learned from it?

First let's consider the issue of speed. Clearly, 31.25 Kbaud will become too slow soon if it is not already. But how fast should it be?

One idea is that it could be any speed. Some UARTs are able to detect the speed of transmission and adapt automatically. This way, speed could grow as needed and as the technology was able to keep up. This argument is curiously reminiscent of arguments to the same effect made by the manufacturers of computer terminals in years past. In the beginning, everyone said, "terminal networks do not have to run any faster than a fast typist can type." So the initial baud rates ranged from 11 to 15 characters per second, comfortably beyond this range and comfortably within prevailing technological limits. However, then came word processors. As text editing programs became smarter at getting the computer to do more work with fewer keystrokes, users started wishing they could see the results more quickly too. The baud rates went up. And they are still climbing. Rates of 9600 baud are now common, with much higher rates not far behind. What drove the increase was not faster typing, but the greater bandwidth required by powerful text and graphics editors to drive the display. To my mind this neatly parallels the situation with MIDI: even if we assume that the current rate is sufficient for performance capture (the analog of typing speed), it will certainly not long be sufficient for computer control of synthesizers (the analogy to driving the display) as users become smarter at getting more music from less performance. The real result of this *laissez faire* attitude in the development of computer terminal networking was that it suffered decades of accretions and kludges as it grew, resulting in a kind of anti-standard. Is this what we want for music?

An interesting alternative was proposed spontaneously on the floor of the MIDISOFT-84 conference, alluded to above, by Guruprem Khalsa. He invited the attendees to consider estimating the bandwidth requirements that will be needed at some point in the future, then identifying some technology which would allow us to achieve that bandwidth now. This would resolve the issue of throughput in a way that would satisfy everyone for that period of time, freeing us to focus on the task of using that bandwidth instead of being drained by

the subtask of retrofitting as data rates evolved. Gordon Moore, one of the founders and current chief executive officer of Intel Corp., presented a formulation that has since become known as "Moore's law" as a way to estimate growth in the electronics industry. It states that for year y, $C \propto 2^{y-1960}$, that is, complexity will double every year from 1960, which was the year he proposed it. If we were to update the year and apply that to MIDI, we might have $C \propto 3125 \cdot 2^{y-1985}$. In 12 years the bandwidth requirement would be on the order of 12 Mbytes/sec. This is a comfortable rate using current networking technology. It can be objected that the technology to run at these rates will be more expensive and for the current purposes of MIDI would not be warranted. Cost is a powerful argument when considering the economies of large-scale manufacturing. But is it ultimately economical to live with a serious flaw that weakens the standard?

Another attendee at the MIDISOFT-84 convention, Charles Goldfarb, questioned the nature of MIDI coding. To explain his position, we need another lesson from the history of word processing and terminal technology. In the beginning of computer terminal technology, the ASCII code (American Standard Code for Information Exchange) was developed with Teletype technology in mind. Some codes were reserved for control of (then state of the art) Teletype machines, such as line feeds, form feeds, carriage motion, and simple communications protocols. It quickly became apparent that document preparation could be done more easily with a macro language embedded in the text that condensed the complicated control codes required to do anything useful into simple mnemonics. Eventually it was noticed that this approach was not device-independent since the mnemonics themselves still referred to device specific actions. Transferring the document to a different printing system required substantial alteration of the document. It became clear that what was really needed was a way to describe page layout in the terms of the formal structure of the document, rather than in terms of the actual structure of the printer, leaving the job of achieving this structure for any given printer to a document compiler. This resulted in the development of high-level document compiler programs such as troff and TEX. Standards for representing document formatting, such as SGML, are emerging to consolidate this area. Goldfarb suggests that we can ask the same thing about MIDI: is its collection of codes targeting the wrong thing?

While this analogy between MIDI and terminal technology is suggestive, it is not complete. By design, MIDI codes represent the performed structure of music. This is a higher level than coding for synthesizer parameters, and a lower level than representing the formal structure of the music. MIDI is to a small degree already device-independent, since the encoded performance does not carry with it a specific meaning as to the sound the resulting synthesizer is to make. It is perfectly adequate for MIDI synthesizers to communicate MIDI codes; a higher level representation is not needed here.

However, the picture changes when we consider how we should represent music made on MIDI synthesizers in a computer memory. Here, we will indeed make the same mistake with MIDI that was made for document preparation unless we develop representations which can express the formal structure of music. A good example of a hierarchical data structure for music can be found in the work of William Buxton (1980). To this end, Goldfarb has recently broached the subject of developing a standard for music databases through ANSI. A meeting was held in Palo Alto in May of 1985 to the end of establishing an ANSI study group to consider this problem.

A problem with MIDI is that it resists attempts to abstract it further. Speaking at a panel on MIDI during the 1984 ICMC in Paris, Buxton pointed out the confusion in MIDI between a network and a data structure. For him, basic to fixing MIDI is separating the structure of the information being communicated from the structure of the network. As it stands, they are inextricably linked, which means that the one can't be changed without the other.

What of the Future?

A corollary of Moore's law states that $\$ \simeq \sqrt{C}$, in other words, cost goes up linearly for exponential

growth in complexity. F. R. Moore has argued (Moore 1981) that while traditional musical instruments are becoming more and more costly to produce, electronic instruments are becoming both more interesting and cheaper. Four years ago he predicted that there would be a point at which these two functions of time would cross, and that there would soon be a time when interesting electronic instruments would be cheaper than traditional instruments. Were this to happen, he argued, it would mean the wide availability of these new instruments, and the serious utilization of them in the music of our culture. Surveying today's popular music scene, it can be easily argued that this time has already arrived. The impact is also evident in schools of music. Projects to explore these new low-cost technologies in many computer music laboratories are underway.

Those who have access to more powerful tools can often be quite caustic about MIDI, seeing only its (numerous) limitations.

> Sooner or later there has got to be a reckoning between musicians and designers of commercially available equipment. Some scientists believe that their mere capacity to respond, in some vague way, to art qualifies them to make judgments about it. This attitude is representative of a terrible cultural problem. Their naive judgments are manifested musically in the sounds and design of commercially available instruments that are just appalling. I shudder to think about the effect on the generation of young listeners who are being exposed to such a low acoustic standard, especially at such high volume levels. I think it's very serious.—Charles Wuorinen (Boulanger 1984).

Irritating as Wuorinen's comments may be to some, he has a point. Engineers and scientists who want to contribute to the technology of music must have a deep insight into the aesthetics of music, otherwise the systems implemented will be archaic and inflexible. Of course, it works the other way too: without deep insight into the means of production, composers and performers will miss the essential contribution which computer technology can make, and the resulting music will be anachronistic.

There are also complaints about the nature of the human interface provided by typical commercial synthesizers.

> A major problem of synthesizers to date, especially recently, is that they constrain the performer to expressing ideas through a limited set of gestures. (Ironically, some electronic instruments from the 1930s to the 1960s were more flexible in this regard.) This "straitjacket" of most "over-the-counter" systems (for example the piano-type keyboard synthesizer), has meant that in many cases, the medium of expression is totally at odds with the musical idea. To follow on this, then, if gesture and idea are tied, and the device is the instrument for capturing the gesture, then the range of input devices could be as diverse as the range of musical ideas.—William Buxton (Appleton 1984).

There are reasons to be happy about MIDI, in spite of what it is, when we consider that it is helping to stimulate research in performance, improvisation, and interactive composition. Many are enthusiastic about using MIDI to help move beyond the limitations of tape music. Most are willing to put up with temporary gestural and sonic limitations to achieve this. Joel Chadabe's work in interactive composition (Chadabe 1984; Chadabe and Meyers 1978), Buxton's work with human interfaces (Buxton 1980), Appleton's work with the Synclavier, and many other projects as well stem from the motive to reclaim performance gesture as a part of the process.

My enthusiasm for this subject is similarly motivated. Throughout my career in electronic and then computer music, little of substance could be done in live performance. Analog synthesizers offered real-time control, but over pathetically limited resources. Software sound synthesis offered tremendous resources, but none of it in real time. Compositions for tape and live instruments usually required the live instrumentalists to synchronize their performance to the tape (unless one chose to declare that synchronization was unimportant, a limited aesthetic option). This made the live performers slaves to the tape part. Now at last we are in a position to make the performer the independent variable and the synthetic part the dependent

variable. A whole continuum of possibilities has opened up. Tape music is at one end, where the electronics ignores the performer. At the opposite end of the continuum are systems that drive synthesizers directly from sensors that extract performance parameters. Here the electronics is slave to the performer. In between are many interesting areas, such as automatic accompaniment (Dannenberg 1984; Vercoe 1984), and numerous other strategies where the electronics and the performer share control. A computer science discipline called control theory (Rouse 1981) is devoted to considering human/machine interaction modalities, static and dynamic systems, and related issues. Suddenly, this seems quite germane to computer music.

Unfortunately, it appears that the needs of the research community will continue to be unaddressed in the marketplace. Most MIDI-based computer systems will continue to be directed at standard applications for nonprogrammers. Many will not even be directed at musicians but will be used for such things as to correct "wrong notes" and provide simple accompaniment for novices.

> It certainly appears inevitable that, when the general-purpose facility gives way to, in the most exalted cases, "work stations," and in the least exalted ones, "glorified organs," the inflexibility that will result will be attended, I fear, by a cessation of imaginative developments, both in terms of the materials of music and in the ways of organizing these materials.—Roger Reynolds (Boulanger 1984).

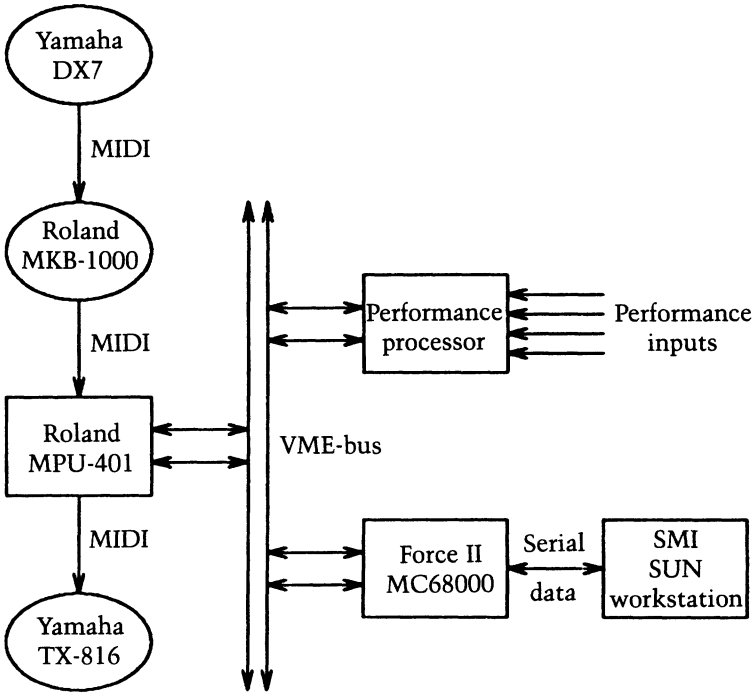
It is clear that we will continue to have to improvise systems that meet our needs from the best available technology. As a result, the field seems destined to be driven by technological imperatives for at least the near future. This has an interesting implication, which can be exposed with the following syllogism: computer music is to computer technology as the steamboat was to steam engine technology, meaning that computer music is still mostly an applied discipline, like the development of steamboats. We are aligned with the historical position of Fulton, rather than Watt. Furthermore, we are at the point in the development curve prior to where steamboats became capable of reliable navigation. We are still mostly engaged in improving the technology to yield the benefits we know are there. Just as reliable worldwide navigation had to wait for the steamboat, many of the experimental research fields awaiting us depend on the availability of appropriate tools. To take other examples, the field of microbiology was only possible after the advent of the electron microscope; astronomy was only possible after the perfection of the telescope. (Another analogy to computer music can be borrowed from the history of astronomy: before the telescope, astronomy was called astrology. After the perfection of the musical equivalent of the telescope, will musicology become musiconomy?)

The musical equivalent of the telescope will probably be a system that combines a special-purpose synthesizer with a general-purpose computer to provide extensible musical data abstractions and operations. While the hard part of systems integration would be accomplished this way, every level of the hardware and software of the resulting system would have to be in the public domain, be easy to modify, and be completely documented.

Fortunately, progress is being made along these lines in many places. One example is the computer music workstation development project at the Computer Audio Research Laboratory. We mean by this term a microcomputer system with the capability of running either out of real time for general-purpose signal processing and composition, or running in real time for performance processing and direct synthesis. In the latter case, the system uses special hardware to do performance capture, analysis, and synthesis. The goal is to extend the range of musical research with such a tool to include performance. Our current prototype uses MIDI devices in combination with other sources and sinks of information (Fig. 10), and it is from this work that I have garnered most of the experience related in this article. This development effort is aimed at providing low-cost tools to the research community that attempt to meet the criteria of those whom I have quoted above (among others). This work would not have been possible prior to the advent of sophisticated, standardized, low-cost, open-architecture components.

The relationship between electronic-instrument manufacturers and the computer music community has been rocky in the past. However, I see the ad-

Fig. 10. Performance laboratory at CARL. Oval components are MIDI devices; squares are processors. Components are: Yamaha DX7 synthesizer, Roland MKB-1000 weighted-action piano keyboard, Yamaha TX-816 synthesizer, Roland MPU-401 MIDI controller, Force II MC68000 VME-bus CPU, Sun Microsystems Inc. SUN workstation. The Performance Processor is an in-house real-time performance processing system under development. These facilities are in addition to our regular timesharing computer resources.



vent of MIDI as the first sign that the commercial synthesizer industry is becoming relevant to the computer music community. By providing low cost, standardized performance processors and synthesizers, our field is gaining tools that will have a broad impact on the kinds of subjects we can investigate and on the numbers of researchers who can participate. In a sense, real-time control research is now the province of anybody with a MIDI synthesizer and a desktop computer. Some (McConkey 1984) even see the fading away of "the distinction between synthesizers and computer music"—meaning, presumably, the distinction between those who use commercial synthesizers and those who have access to facilities of computer music research centers. Perhaps so, but there will always be a distinction between making music and doing musical research, even if the latter is in the form of musical compositions that use commercial synthesis systems. The research uses of computer music tools—both scientific and musical—will always lead commercial application. The development of MIDI signals the emergence of several important technologies from the laboratory and into the field. If the intercommunication between the synthesizer industry and the computer music community can grow, as I see happening all around me, it presages better things to come.

References

Appleton, J. 1984. "Live and in Concert: Composer/Performer Views of Real-Time Performance Systems." *Computer Music Journal* 8(1):48–51.

Bischoff, J., R. Gold, and J. Horton. 1978. "Music for an Interactive Network of Microcomputers." *Computer Music Journal* 2(3):24–29. Reprinted in C. Roads and J. Strawn, eds. 1985. *Foundations of Computer Music.* Cambridge, Massachusetts: MIT Press.

Boulanger, R. 1984. "Interview with Roger Reynolds, Joji Yuasa, and Charles Wuorinen." *Computer Music Journal* 8(4):45–54.

Buxton, W. 1980. "A Microcomputer-based Conducting System." *Computer Music Journal* 4(1):8–21.

Chadabe, J. 1984. "Interactive Composing: An Overview." *Computer Music Journal* 8(1):22–27.

Chadabe, J., and R. Meyers. 1978. "An Introduction to the PLAY Program." *Computer Music Journal* 2(1): 12–18. Reprinted in C. Roads and J. Strawn, eds. 1985. *Foundations of Computer Music*. Cambridge, Massachusetts: MIT Press.

Dannenberg, R. 1984. "An On-Line Algorithm for Real-Time Accompaniment." In W. Buxton, ed. *Proceedings of the 1984 International Computer Music Conference*. San Francisco: Computer Music Association.

International MIDI Association. 1983. *MIDI Musical Instrument Digital Interface Specification 1.0*. North Hollywood: International MIDI Association.

Mathews, M., and F. R. Moore. 1970. "GROOVE—A Program to Compose, Store, and Edit Functions of Time." *Communications of the Association for Computing Machinery* 13(12): 715–721.

McConkey, J. 1984. "Report from the Synthesizer Explosion." *Computer Music Journal* 8(2): 59–60.

Moore, F. R. 1981. "The Futures of Music." *Perspectives of New Music* 19(1): 212–226.

Rouse, W. B. 1981. "Human-Computer Interaction in the Control of Dynamic Systems." *ACM Computing Surveys* 13(1): 78–91.

Vercoe, B. 1984. "The Synthetic Performer in the Context of Live Performance." In W. Buxton, ed. *Proceedings of the 1984 International Computer Music Conference*. San Francisco: Computer Music Association.

Appendix

In the following descriptions, the values in parenthesis indicate arguments to the command.

Channel Commands

key off (*key, velocity*)—turns off one voice playing *key*. *Velocity* is used to determine the characteristic of decay.

key on (*velocity, key*)—turns on one voice with pitch *key*. *Velocity* is used to determine the characteristic of attack. In *poly* mode, sequential *key on* commands create chords. In *mono* mode, if a voice was sounding, a new *key on* command will simply shift pitch to the new key, providing the mechanisms for a legato effect. As a special case, a *key on* command with a velocity of 0 is a form of *key off*. Key indices encode equal-tempered semitones. Middle C is key index 60. Five-octave keyboards range from 36 to 96. Eighty-eight-note keyboards range from 21 to 108. Default velocity, in the absence of pressure sensors, is 64.

polyphonic key pressure (*key, pressure*)—encodes key bottom pressure change for the indicated key on the indicated channel. It is sent by keyboards that discriminate individual key bottom pressure; it is recognized by polyphonic synthesizers capable of responding to individual key pressure changes.

control change (*index, value*)—supplies a new value for indexed performance controllers, such as modulation wheels, joysticks, switches, pedals, etc., but not pitch benders. The upper range of indexes are reserved for mode-setting commands. (See Table 2.)

program change (*index*)—selects a timbre or synthesis technique.

channel pressure/after-touch (*pressure*)—pressure change affects all voices responding to that channel.

pitch bend (*LSB, MSB*)—encodes a 14-bit pitch bend quantity from a pitch wheel. The value of 8192 corresponds to the center detent. The actual range of pitch change for any value of pitch bend is defined in the synthesizer.

System Exclusive Command

system exclusive (*manufacturer ID number*)—sends manufacturer-specific synthesizer information, such as patches, parameters, functions, and messages. Manufacturer ID numbers are assigned by the International MIDI Association. Information following the ID number is manufacturer-specific. This command is terminated by the *end of block* command or by starting any other command. System exclusive commands are nonpreemptive.

System Common Commands

These commands are for sequencer control. For "sequencer" one can also read "computer."

position select (*MSB, LSB*)—select starting position within a sequence, specified by a 14-bit index.
program select (*index*)—select a sequence by its *index*.
tune request—adjust tuning of synthesizers.
end of system exclusive—the official way to terminate a system exclusive command.

Real-time Commands

Real-time commands can preempt all commands including *system exclusive*, if necessary, to preserve timing.

timing clock—emitted 24 times per quarter note by a master sequencer to synchronize time bases between multiple sequencers. At *MM* = 60, a quarter note equals 1 sec.
timing clock with measure end—sent instead of *timing clock* at the end of measures.
start—start sequencers, begin sending *timing clocks*.
continue—continue sequence from where it was stopped on the last *timing clock*. Begin sending *timing clocks*.
stop—stop sequencers. The *stop* command is then continuously sent in substitution for *timing clock* to maintain synchronization. This allows for "lead-in" and "punch-in" to be synchronized.
active sensing—an "I am alive" signal emitted a few times a second (no less than every 320 msec, according to the specification) to indicate the continued functioning of a transmitting MIDI device. Receiver should stop synthesis if *active sensing* commands stop coming. Note: active sensing commands are preempted by all other MIDI commands. Thus, they cannot be relied on for timing information.
system reset—reset to power-on condition.