SESSION 11

INFORMAL PRESENTATIONS: TOOLS

Moderator—Sarah E. Ransdell, New College

MIDI and Macintosh: Searching for a better mousetrap

JAMES M. KIELEY

Pitzer College, The Claremont Graduate School, Claremont, California

This paper will explore how MIDI (musical instrument digital interface) technology can benefit researchers interested in using Apple Macintosh computers for stimulus control and response timing. One example presented here will illustrate how MIDI equipment can be adapted to provide millisecond precision response timing that cannot be obtained with solely software-based timing techniques. Another example will show how to use MIDI to control sound-generating devices that far surpass the sound-producing capabilities built into the Macintosh. There will also be a discussion of how certain MIDI equipment can provide information on the velocity and the duration of a response that is not available with traditional laboratory hardware. The examples presented here rely on a commercial software library called MIDIBASIC that was designed to be used with Microsoft QuickBasic. The availability of a millisecond timer that can be called from QuickBasic may offer a special advantage for researchers who have previously written programs in an Apple II environment using AppleSoft Basic.

Many psychologists have been attracted to the idea of using the Apple Macintosh computer for conducting experiments, largely because of its powerful graphics capabilities and, to a lesser extent, its built-in ability to reproduce simple sounds and synthesized speech. The recent introduction of more affordable models with powerful 32-bit microprocessors, better sound, and color monitors provides further incentive for using the Macintosh in research. Yet there are potential limitations and stumbling blocks for anyone who attempts to use a Macintosh for data collection. In this paper, we will describe how MIDI can be used to help overcome some of these limitations, and we will also suggest ways in which MIDI can be used to collect data that are unavailable with more traditional research equipment.

What is MIDI?

MIDI stands for "musical instrument digital interface." MIDI is not a particular piece of hardware, but rather a set of specifications for communication that was first introduced in 1982 by a group of electronic musical instru-

ment manufacturers (Boom, 1987). The medium for MIDI communications is a serial line that operates at 31.25 k/sec. MIDI information is processed in the form of MIDI messages that fall into one of five different classes of data. This information is typically exchanged between MIDI devices such as electronic keyboards and computers. MIDI IN, MIDI OUT, and MIDI THRU ports linked through appropriate cabling provide a physical connection between MIDI devices. At least one type of port is built into any MIDI musical instrument. MIDI ports for a computer must be provided by an additional piece of hardware called a *MIDI interface*.

MIDI equipment can be set up in a wide variety of configurations, which can be classified into two broad categories. One includes configurations in which two or more MIDI musical instruments (e.g., two synthesizers) are set up so that the input mechanism (e.g., keyboard) on one instrument controls the sound output on another. The other consists of configurations in which one or more MIDI musical instruments are attached through a MIDI interface to a microcomputer. The microcomputer in this case usually has sequencing software available that allows MIDI information output from the MIDI musical instrument to be captured digitally. The information can be held in RAM or written to disk. Most MIDI sequencing soft-

Correspondence should be addressed to James M. Kieley, Pitzer College, The Claremont Graduate School, Claremont, CA 91711.

ware can record 16 separate tracks of MIDI data. Each track of digital data can be individually edited. Each track can provide real-time control for one or more MIDI instruments during later playback. During playback, the computer and the MIDI musical instrument function together as a sort of sophisticated player piano. It is also possible and relatively straightforward to write computer programs that collect data from or provide control over MIDI devices. All that is needed is a small library of MIDI functions written in assembly language that can be called from a high-level language such as QuickBasic, Pascal, or C.

Millisecond Timing on the Macintosh

One limitation frequently encountered with any Macintosh model is the difficulty of accurate response timing. Software-based timing routines are available that, under appropriate circumstances, will time with millisecond precision (Kieley & Higgins, 1989; Powell & Gillespie, 1986; Rensink, 1990; Westall, Perkey, & Chute, 1986, 1989), yet the functionality of the software-based timer may be affected by the particular machine and operating system version with which it is used. This is complicated by the fact that Apple continues to make changes in firmware and in the operating system as new Macintosh models are introduced. A further problem is that the successful implementation of a software millisecond timer may depend on the availability of a programmer proficient in low-level assembly language and almost always requires a programmer proficient in a relatively complex high-level language such as Pascal or C.

The software-based millisecond timers currently available for the Macintosh operate either by directly accessing one of the timers on the computer's VIA (versatile interface adapter) chip or by making a call to the TIME-MANAGER ROM routine. Each method of timing has its advantages and disadvantages, as has previously been documented by Westall et al. (1989). The accuracy of a VIA millisecond timer may be greater, but only a TIME-MANAGER timer will allow the internal sound-generation capabilities of the Macintosh to be used while the timer is concurrently running. The routines presented by Westall et al. (1989) can be used for VIA timing, whereas either the Westall et al. (1989) routines or those presented by Rensink (1990) can be used for TIME-MANAGER timing. The millisecond timers presented by Powell and Gillespie (1986) and Westall et al. (1986) may cause system crashes, and the millisecond timer presented by Kieley et al. (1989) is subject to timing errors with Version 6.0 and later of the Macintosh operating system.

Another aspect of obtaining accurate timing on a Macintosh that no software solution alone can address is the variable delay that can occur between the time when a mouse or key click is initiated and when the timer-halting event is registered. With early Macintosh models (i.e., Macintosh Plus and earlier), the accuracy of a timer triggered from the keyboard is always limited, because keypress events are polled only every 16 msec during the screen's vertical retrace interrupt. A similar variable delay occurs

if the ROM Toolbox routine `BUTTON` is used to detect whether the mouse button has been pressed. However, with the early models, it is also possible to tell whether the mouse button is down by directly reading the VIA, which theoretically permits ± 1 msec timing accuracy. Details on how to poll the VIA in this manner are discussed in Powell and Gillespie (1986) and Algarabel, Sanmartin, and Ahuir (1989). On the more recent Macintosh models (SEs and Mac II series), consistent millisecond precision timing cannot be obtained without the use of an external response-triggering mechanism. This is because on later models the keyboard and the mouse communicate to the CPU through the VIA by means of the Apple Desktop Bus (ADB), which is an asynchronous serial communication bus that includes its own microprocessor. Button or keypress events on ADB models are subject to a variable delay of up to 11 msec.

One approach to dealing with the timing error that results from the limitations of hardware is to attach an appropriate external box to a serial port on any Macintosh or to install a suitable card in a Macintosh with an available internal slot. These "black box" solutions can provide an interface with millisecond timing precision to a response-triggering mechanism such as a telegraph key or a voice-activated relay (VAR), but sometimes at a considerable expense. The cost of providing a commercially produced "black box" for each machine in a laboratory of several Macs can be substantial (approximately \$600-\$2,000 for each machine). A less expensive alternative would be to design and fabricate a "black box" from parts, but this would require a skilled technician with a degree of engineering ability. Another expense that would accompany either a commercial or a fabricated "black box" consists of programming overhead. Typically, a "black box" is controlled through a library of routines written in assembly language that must be called from a high-level language such as Pascal or C. To complicate matters, in some instances, the library routines for a given "black box" are unusable with the updated ROM chips and operating systems that accompany the new Macintosh models.

From this discussion, it follows that a useful piece of equipment for researchers interested in using Macintosh computers to conduct reaction time experiments would be a precision interface that could be used with a variety of input devices. The interface should require a minimal investment in terms of add-on hardware costs and programming time. Software libraries should be available to access the interface from a variety of programming languages. The libraries should be updated regularly to maintain compatibility with the entire Macintosh line. The device itself should operate fast enough so that it can provide millisecond resolution without variability. MIDI equipment used with appropriate software can easily be adapted to meet each of these criteria.

MIDI Timing and MIDI Messages

The MIDI-based timer response-triggering chain shown in Figure 1 consists of four stages. The first two stages

Figure 1. Four-stage MIDI-based response-triggering chain.

consist of a hardware trigger and a device that detects a voltage drop caused by the closing of the contacts on the hardware trigger and then converts that event into MIDI information. These first two stages can consist of discrete devices or of a single unit that combines both functions. Examples of discrete Stage 1 devices are traditional response mechanisms such as a telegraph key or a microphone attached to a voice-activated relay.¹ The Stage 2 device that would need to be combined with either of these devices would be a unit that converts a voltage drop caused by the closing of two contacts on a mechanical switch into MIDI information. One way to obtain this conversion is to perform a simple modification on an inexpensive MIDI keyboard (see Figure 2). Another method—given that the appropriate engineering skills are available—would be to build a simple MIDI circuit like that discussed in Chamberlin (1985).

An unmodified MIDI keyboard can combine the functions of Stages 1 and 2 for situations in which a keyboard keypress is an acceptable response mechanism. In this application, it is best to use a keyboard of at least moderate quality, because the keys on an inexpensive keyboard (typically those retailing for less than \$300) may begin to make inconsistent contact after long use, so that accurate data collection becomes impossible.

Some care should be taken in selecting a device to provide the Stage 2 voltage-to-MIDI conversion, because the speed with which the conversion can take place may vary under different information loads. The microprocessors found in even inexpensive MIDI musical instruments manufactured in the last few years have been relatively

fast, so that CPU delay error should be minimal in any situation in which only the time taken to register an initial response is of concern to the experimenter. On the other hand, delays can be greater when several responses are made in close succession and the experimenter is interested in separately recording the time taken to make each response.

The third stage of the response-triggering chain shown in Figure 1 is the MIDI interface unit, which provides MIDI ports for the computer; the fourth stage is the computer itself. Under normal circumstances, neither of these devices will contribute variable delay to a response-triggering chain.

MIDI information exchanged at the rate of 31.25 K/sec is in the form of 8-bit bytes, which means that about 3 bytes of data can be sent every millisecond.² As noted earlier, the MIDI communication protocol consists of five different types of information or messages. A Channel Voice Message is the type that is the most relevant when MIDI technology is used for data collection. One type of Channel Voice Message that will be used in the examples presented in this paper is a Note On message. A Note On message generated by a MIDI instrument can be used to trigger a millisecond timer in a laboratory situation. A Note On message has three 1-byte parts: a status byte, a note byte, and a velocity byte. The status byte is a number between 144 and 159 (decimal) that indicates which of the 16 available MIDI-channels has been activated.³ The note byte is a number between 0 and 127 that represents the pitch of a note that was generated. The velocity byte is a number between 0 and 127 that indicates the speed with which a key has been pressed. It takes MIDI about .96 msec to transfer a 3-byte Note On sequence; this is the degree of resolution possible with a MIDI-triggered timer. Under normal circumstances, the signal generated by a Note On instruction is terminated by a second Note On instruction that specifies a zero value for velocity.⁴

Two other types of Channel Voice Messages that will be relevant in the examples that follow are Program Change messages and Channel Pressure messages. Program Change messages consist of 2 bytes of information. Similar to a Note On message, a Program Change message has a status byte represented by a number between 192 and 207 that indicates which of the 16 MIDI channels an instruction will be sent for. It also has a data byte that is a number between 0 and 127 that indicates which of the available memory patches (i.e., sound programs) on a sound generator should be loaded. Channel Pressure or aftertouch messages consist of a status byte, which is a number between 208 and 223 that indicates the MIDI channel to be selected, and a data byte, which is a number between 0 and 127 that indicates how much pressure is being applied to a key after it has initially been pushed down.

A MIDI-Macintosh Laboratory

The arrangement shown in Figure 3 represents the setup used in a functional MIDI/Macintosh experimental psychology laboratory. The hardware consists of an Apple

Figure 2. Modification that allows a Casio MT-240 to convert off/on output from a voice-activated relay to a MIDI signal that can be passed on to a MIDI interface unit attached to a microcomputer.

Macintosh Plus computer, an Opcode Professional Plus MIDI interface unit, a Lafayette voice-activated relay, a Shure SM-58 microphone, a Casio MT-240 MIDI keyboard, a Yamaha TX-81Z FM tone generator, a Fostex 6301B personal monitor (i.e., a small external speaker with a built-in power amplifier), and two MIDI cables. A number of alternatives could be used as functional equivalents for any component in this list. The MIDI-specific equipment consists of the interface and the two cables, which can be purchased for under \$100.

The example applications described below were designed to work with a software library called MIDIBASIC (Version 3.0), which is marketed by Altech Systems (list price, \$99). This library of 13 routines can only be accessed from MicroSoft QuickBasic for the Macintosh. Although ease-of-programming considerations led to the decision to use MIDIBASIC examples in this paper, Altech Systems also markets MIDIPASCAL (Version 3.0), which can be adapted to uses similar to those described here by Think Pascal or Think C programmers. According to Altech Systems, MIDIBASIC is compatible with any

Macintosh Plus, Classic, SE, SE/30, II, IIfx, IICx, IICi, or IISI with at least 1 MB RAM. The Macintosh IIfx reportedly has problems with many MIDI devices. The Macintosh Portable may have problems with certain MIDI applications, because when the computer is powered by its battery pack and no serial port activity is detected, the serial ports will shut down to save energy. The early Macintosh models (i.e., pre-Macintosh Plus) can be used to run compiled MIDIBASIC problems, but memory limitations would make them inappropriate for program development. The examples presented here were tested on both a Macintosh Plus and a Macintosh IICx.

MIDI Timing Applications

An example that illustrates how MIDIBASIC can be implemented as a simple millisecond precision timer is shown in Listing 1. In this example, the MIDIBASIC library installs a VIA-based timer that is triggered by the Macintosh's keyboard (or alternatively the mouse button). This implementation of MIDIBASIC acts as a functional equivalent to the routine presented by Westall et al. (1989)

Figure 3. Equipment used in a functional MIDI/Macintosh experimental psychology laboratory.

operating in the VIA mode. However, there are two potential advantages to the use of the MIDIBASIC library routine. One is that the library can be accessed easily from BASIC. This means that programmers who have migrated to the Macintosh from the Apple II/Applesoft BASIC environment and who do not know Pascal or C can quickly put together programs that require precise timing with little relearning. The other advantage is that MIDIBASIC is a commercial product that has been purchased by many musicians and hobbyists. This popularity provides a motivation for the authors to provide frequent updates so that the library will function properly with as many Macintosh models as is possible.

Since the program shown in Listing 1 is written so that the computer's keyboard or the mouse button serves as a response triggering mechanism, this application is subject to the hardware-specific variable measurement error that was discussed previously. This particular use of MIDIBASIC would be appropriate for a situation in which equipment setup must be simplified and the experimental power is expected to be large.

Taken from Listing 1, `LIBRARY "MIDIBASIC"` installs the MIDIBASIC routines so that they can be called from

a BASIC program running in interpreted mode. Optionally, to save memory, specific MIDIBASIC routines can be installed as MBLC resources either into a program or into QuickBasic itself. Details on how to do this are included in the MIDIBASIC manual.⁵ `MIDIOPEN 100,100` sets up input and output communications buffers that are 100 bytes each in size (these values are somewhat arbitrary). `MIDITIME 0` initially turns off timestamping. `MIDITIME Ticklength%` (where `Ticklength%` is set equal to 782) turns on timestamping and tells the MIDIBASIC timer to divide time into intervals of 1-msec duration. Simple response timing with MIDIBASIC can be carried out by using the commands `SETTIME` and `GETTIME`. From Example 1 in Listing 1, `SETTIME` is set to the value of `ticks&` (which has previously been set equal to zero) immediately before the stimulus is presented. As soon as a response is made (in this case by pressing the space bar), the command `GETTIME` reads how many milliseconds have elapsed since the timer was reset at zero.

The code that is specific to Example 2 in Listing 1 illustrates the changes that are necessary to allow millisecond timing from an external response-triggering mechanism (e.g., a voice-activated relay) connected to

Listing 1

**Implementation of MIDIBASIC As a Millisecond Precision
Timer Using Either the Macintosh Keyboard (Example 1)
or an External Switch That Communicates to the
Computer Through MIDI Hardware (Example 2)
As a Response-Triggering Mechanism**

```

REM *****
REM ** This listing provides examples of how to implement the MIDIBASIC **
REM ** millisecond timer using either the Macintosh's keyboard or the **
REM ** mouse button as the response triggering mechanism (Example 1) **
REM ** or a response triggering mechanism that communicates to the **
REM ** Mac with the assistance of add-on MIDI hardware (Example 2) **
REM *****

LIBRARY "MIDIBASIC 3.0": REM Load library
MidiOpen 100,100: REM Set up input and output buffers
Ticklength%=782: REM Set Ticklength% equal to 1 millisecond
MidiPort 1: MidiPort 3: REM 1 MHz communication rate using modem port
MIDI 0: REM Reset and enable input/output buffers
MidiTime 0: REM Initially turn timestamping off
MidiTime Ticklength%: REM Turn timestamping on
GOSUB BlankScreen: REM Clear screen including menu bar
CALL MOVETO(250,147): PRINT "+": REM Put up warning signal on screen
x%=30: GOSUB Delay: REM Wait for 500 milliseconds
CLS: REM Clear screen

RANDOMIZE TIMER
x%=INT(60*RND)+30: REM Set random delay equal to 500-1500 milliseconds
GOSUB Delay: Wait random delay period
REM At this point the program should GOSUB Example 1 if the Macintosh's
REM own keyboard will be used as a response triggering mechanism or to
REM Example 2 if a triggering mechanism will be routed to the Mac thru MIDI
REM hardware.

GOSUB Example 1
GOSUB Example 2
CALL GETTIME(ticks&)
RT(I)=ticks&: REM Read the value of ticks& into another variable for storage

Example1:
WHILE INKEY$<>"":WEND: REM Clear event queue
CALL MOVETO(250,132)
A&=PEEK(&H16A)+1: REM Make sure target always starts to be drawn on
WHILE PEEK(&H16A) < A&: REM the screen at the same relative point in
WEND: REM time by waiting for the vertical blanking interrupt to occur
ticks&=0: SETTIME ticks&: Reset the millisecond timer to zero
PRINT "<<<TARGET>>>": REM Put target up on screen
WHILE INKEY$<>" ":WEND: REM Wait for space bar to be pressed
RETURN

Example2:
REM: Call the next line only once
MIDI PORT 1: MIDI PORT 3: MIDI 0
REM Select the appropriate line from the next two lines to make sure that
REM perviously generated response can't falsely trigger the timer
MIDI 5: REM Clear out MIDI input buffers
CALL MOVETO(250,132)
A&=PEEK(&H16A)+1: REM Make sure target always starts to be drawn on
WHILE PEEK(&H16A) < A&: REM the screen at the same relative point in
WEND: REM time by waiting for the vertical blanking interrupt to occur
ticks&=0: SETTIME ticks&: Reset the millisecond timer to zero
PRINT "<<<TARGET>>>": REM Put target up on screen
CALL MIDIIN%(z%,ticks&): REM Wait for MIDI data to be input
WHILE z = -1:CALL MIDIIN%(z%,ticks&): WEND: 'Example 2'
RETURN

Delay:
A&=PEEK(&H16A)+x%
WHILE PEEK(&H16A) < A&:WEND
RETURN

BlankScreen:
VSize%=PEEKW(PEEK(&H904)-116)
HSize%=PEEKW(PEEK(&H904)-114)
setrect v%(0),0,0,HSize%,VSize%
Gray&=PEEK(&H9EE)
CALL SetRectRGN(Gray&,0,0,HSize%,VSize%)
WINDOW 1,,(0,0)-(HSize%,VSize%),3
RETURN

```

the Macintosh through peripheral MIDI equipment (i.e., the Stage 2 converter and the Stage 3 MIDI interface discussed previously). This example has been tested successfully with a modified Casio MT-240 keyboard (see Figure 2) serving as the Stage 2 converter. MIDI PORT 1 sets the interface speed to 1.0 MHz (optional values are .5 MHz and 2.0 MHz). It is important that this speed match the speed of the selected MIDI interface unit that will be used. MIDI PORT 3 specifies that the computer's modem port will be used for MIDI communications (the printer port may be used as an option). MIDI 0 clears and enables the MIDI input/output buffers.

When the timing of responses is triggered from external MIDI hardware, MIDIIN is used to accept information from the input buffer. MIDIIN can be used in either of two ways. When timestamping is turned off with MIDITIME 0, the format for a MIDIIN command is MIDIIN midibyte%, where midibyte% is a variable representing the value of the MIDI data. When timestamping is turned on with MIDITIME Ticklength% (Ticklength% is set equal to 782 or some other nonzero number), the format for a MIDIIN command is MIDIIN (midibyte%,ticks&), where ticks& is the current time relevant to the last SETTIME instruction. The WHILE:WEND loop checks to see whether or not the value of midibyte% is equal to -1. A value of -1 indicates that the input buffer is empty and that no response has been made yet. In this application, the first byte (i.e., the status byte) of a Note On message will stop the timer. The value for a Channel 1 Note On status byte is 144, so as soon as midibyte% changes from -1 to 144, the timer-halting response will be registered. MIDI 5 is called before each stimulus presentation to clear the input buffer (i.e., reset midibyte% equal to -1) to prevent false triggerings.

Since MIDIBASIC uses a VIA-based timer to achieve the maximum possible timing accuracy, it is not possible to use the Macintosh's internal sound-generating capabilities while the timer is installed and running. Far from being an insurmountable problem, this limitation makes it possible to demonstrate the true power of MIDI. MIDI was designed to control external sound-generating instruments that exceed the minimal sound producing capabilities built into the Macintosh. The entire repertoire of performance characteristics of any instrument with a MIDI IN port can be controlled while the computer simultaneously times responses. Listing 2 can be used to control a Yamaha TX-81Z. MIDIOUT (midibyte%,ticks&) is used to output a MIDI byte and its timestamp. The first two MIDIOUT instructions send out values of 192 and 99, which indicate that the program on Channel 1 should be changed to program number 99, which happens to be called "Funky Pick" on the TX-81Z.⁶ The next group of three MIDIOUT instructions send out Note On values of 144, 60, and 64, which communicate to the TX-81Z "over Channel 1," "play middle C," "with an attack velocity of 64." The third group of MIDIOUT instructions indicates that the middle C controlled on Channel 1 should be turned off

Listing 2
Example Showing How MIDIBASIC Can Be Used to
Control a MIDI Electronic Musical Instrument
to Generate Auditory Stimuli

```

REM *****
REM ** This listing provides a simple example of how to use MIDIBASIC **
REM ** to control an external MIDI sound generation device, in this case **
REM ** a Yamaha TX-81Z FM tone generator **
REM *****
LIBRARY "MIDIBASIC 3.0" : REM Load library
MidiOpen 100,100 : REM Set up input and output buffers
Ticklength%=782 : REM Set Ticklength% equal to 1 millisecond
MidiTime 0 : REM Initially turn timestamping off
MidiTime Ticklength% : REM Turn timestamping on
progchan%=192 : REM Set program change channel select variable to MIDI channel 1
program%=99 : REM Set program change variable equal to 99
midiout progchan% : REM Change program on channel 1
midiout program% : REM to program #99 on TX-81Z, "Funky Pick"
channel%=144 : REM Set Note On channel select variable to MIDI channel 1
note%=60 : REM Set Note On note variable to 60 which is middle C
velocity%=64 : REM Set Note On velocity variable to 64 which is moderate
midiout channel%,ticks& : REM Send Note On message on channel 1
midiout note%,ticks& : REM so that a middle C is played
midiout z%,ticks& : REM with a moderate attack velocity
ticks&=0 : SETTIME ticks& : Reset the millisecond timer to zero
GETTIME ticks&
WHILE ticks& < 151 : GETTIME ticks& : WEND : REM Wait 150 milliseconds
velocity%=0 : REM Set Note On velocity variable to 0 which means no sound
midiout channel%,ticks& : REM Send Note On message on channel 1
midiout note%,ticks& : REM to turn off the middle C note
midiout velocity%,ticks& : REM that was previously started

```

because a velocity value of 0 indicates that a Note On should have no volume. Far more complex MIDIOUT command sequences are possible. In theory, MIDI can send 1,000 Note On/Note Off instruction sequences per second, which equates to 500 notes.

Other Applications

The previously described MIDI applications for the Macintosh are for the most part analogous to ways in which microcomputers are commonly set up to be used in psychological research laboratories at present. The special characteristics of certain types of MIDI hardware also suggest that MIDI can be applied to provide stimulus control and data-gathering capabilities that are not available with more commonly used laboratory equipment.

Although the designers of MIDI primarily intended it to be used by musicians to control multiple musical instruments from a single source, over the past few years, the variety of applications for the protocol have expanded. In the entertainment industry, MIDI has been used to control tape recorders, and it also has been interfaced to SMPTE time code devices so that it can be used in film and video production work. Others have devised interfaces that allow MIDI to control lights and simple robots. The fact that it is a well-documented protocol that has already been adapted in a relatively wide range of applications suggests that MIDI may be useful in a variety of other instrument control situations, including some that may be of interest to research psychologists. For example, a Macintosh connected to appropriate MIDI equipment can be used to control a slide projector or other light source during an experiment in which response times are also collected on the same computer.

Generally speaking, when experimenters collect data for a reaction time experiment in which the specified response is some form of motor movement (e.g., a keypress), they are limited to recording as accurately as possible the time taken to initiate a response. Information about the time that has elapsed from when the response was first begun until it was completed is unavailable. Another way of thinking about this is to note that when a response key is pressed, there is usually some specific force that must be applied or an activation threshold that must be exceeded before a response can be registered. No data are available for subthreshold movements that might have indicated the time that elapsed between when a motor movement first occurred and when it reached maximum deployment or extension. This type of data could be of interest in applications in which the experimenter is trying to measure the uncertainty that might be associated with a given response across different levels of difficulty. It is reasonable to suggest that a slower motor movement (i.e., the time difference between the beginning and the completion of a response) might reflect a relatively larger degree of uncertainty in a complex as compared to a simple decision condition. The attack-velocity parameter provided by some MIDI keyboards is capable of providing a measure of this sort.

Although the attack-velocity byte is part of a Note On message produced from any MIDI instrument, the value represented by this byte is sometimes meaningless. Due to their mechanical design, very inexpensive keyboards generally are incapable of producing meaningful information about how fast a key is pressed. In these cases, the attack-velocity-byte output by the keyboard is assigned an intermediate value of 64 (see Example 1 in Listing 3). More sophisticated keyboards often have an upper and a lower stationary contact point within each key. When a key is depressed, the time taken for a third moving contact to travel from the upper stationary contact to the lower stationary contact can be interpreted as the attack velocity. In musical applications, attack velocity can be used to control a number of parameters on a sound generator such as volume, brightness, and pitch bend. In laboratory situations, the same MIDI information may be able to provide valuable data about the nature of a response. Example 2 in Listing 3 shows output from a keyboard with attack-velocity capability.

The most sophisticated MIDI keyboards provide yet another type of information called aftertouch. Aftertouch is produced by a pressure sensor underneath the key bed. The last five lines of output shown in Example 3 of Listing 3 are examples of aftertouch data. These lines represent the amount of pressure applied to a key after it has first been pressed. The value 208 indicates that the aftertouch data is for MIDI Channel 1. The five descending values (i.e., 5, 4, 2, 1, 0) indicate that the pressure is decreasing over time. The longer a key is held, the longer the stream of aftertouch data that will be produced. One application for this sort of information would be a divided attention situation in which overlapping stimulus presentations and responses are anticipated. The aftertouch

Listing 3

Output Generated from Various Types of MIDI Keyboards

```

REM *** Example 1 ***
REM .....
REM *** The two rows of numbers in Example 1 are a data dump from a ***
REM *** keyboard that is not velocity sensitive. The first row indicates that ***
REM *** over MIDI channel 1 a middle C key was pressed with an attack ***
REM *** velocity of 64 which is the 'dummy' value assigned when attack ***
REM *** velocity data is not available. The second row indicates ***
REM *** that the middle C is turned off. ***
REM .....
144 60 64
144 60 0

REM *** Example 2 ***
REM .....
REM *** The two rows of numbers in Example 2 are a data dump from a ***
REM *** keyboard with attack velocity, but no aftertouch. The first row ***
REM *** indicates that over MIDI channel 1 a middle C key was pressed ***
REM *** with an attack velocity of 82 and the second row indicates the ***
REM *** note is turned off. No information is available regarding for how ***
REM *** long the key press or response was sustained. ***
REM .....
144 60 82
144 60 0

REM *** Example 3 ***
REM .....
REM *** The seven rows of numbers in Example 3 are a data dump from a ***
REM *** keyboard with aftertouch. The first row indicates that over MIDI ***
REM *** channel 1 a middle C key was pressed with an attack velocity of ***
REM *** 100. The second row indicates the note is turned off. The last ***
REM *** five rows of aftertouch data indicate that the key was gradually ***
REM *** released after it was first pressed. ***
REM .....
144 60 100
144 60 0
208 5
208 4
208 2
208 1
208 0

```

data from a response to one presentation might be of value in understanding how the duration of an initial response might interfere with the stimulus processing and response to a simultaneous or immediately following presentation.

MIDI could also be of particular value when data are collected from several individuals who are tested simultaneously. The Casio keyboard modification for a single input device shown in Figure 2 can easily be extended to allow each of several input devices to be assigned to a given key. In an experimental situation, this means that several individuals could each have one or more mechanical response devices, such as a telegraph key, assigned to a different note represented by a key on a keyboard. The fact that a MIDIBASIC MIDIIN command can capture and timestamp Note On messages means that response time data can more or less simultaneously be collected from several subjects. The reaction time data for a given subject would be identifiable from the Note On note byte associated with a given reaction time. The note byte would identify which response key associated with a given note on the keyboard had been pressed. Obviously, in this sort of application, timing precision would suffer a bit because brief delays would be inevitable as the number of simultaneous respondents grew. However, in many group testing situations in which large effect sizes are expected, it would be possible to use a single Macintosh to control stimulus presentations (e.g., generated from a projector

or a MIDI controlled sound system) and also collect response data.

Conclusion

The examples presented here show how MIDI can be used to improve the use of Macintosh computers in experimental psychology applications. MIDI can be teamed with inexpensive external hardware to provide response timing accuracy comparable to that obtainable on other personal computers such as the Apple II or the IBM PC. The fact that hundreds of thousands of MIDI musical instruments are sold each year throughout the world attests to the fact that MIDI is a well-established standard that draws support from a wide variety of available peripheral products. Some of these can be adapted for laboratory use. Accessing MIDI with a product like MIDIBASIC provides programmers who have previously worked in a BASIC environment on Apple IIs an easy transition to take advantage of the improved speed, memory capacity, and graphics that are available on the Macintosh. In addition to providing a method to conduct response timing with true millisecond precision, MIDI equipment can be used to capture other types of response data that are not available with traditional laboratory equipment.

REFERENCES

- ALGARABEL, S., SANMARTIN, J., & AHUIR, F. (1989). A voice-activated key for the Apple Macintosh computer. *Behavior Research Methods, Instruments, & Computers*, 21, 67-72.
- BOOM, M. (1987). *Music through MIDI*. Redmond, WA: Microsoft Press.
- CHAMBERLIN, H. (1985). *Musical applications of microprocessors*. New York: Hayden.
- KIELEY, J. M., & HIGGINS, T. S. (1989). Precision timing options for the Apple Macintosh family of computers. *Behavior Research Methods, Instruments, & Computers*, 21, 259-264.
- POWELL, D., & GILLESPIE, S. (1986). *Millisecond timer mini-driver* [Computer program]. Portland, OR: Reed College.
- RENSINK, R. A. (1990). Toolbox-based routines for Macintosh timing and display. *Behavior Research Methods, Instruments, & Computers*, 22, 105-117.
- WESTALL, R. [F.], PERKEY, M. N., & CHUTE, D. L. (1986). Accurate millisecond timing on Apple's Macintosh using Drexel's MilliTimer. *Behavior Research Methods, Instruments, & Computers*, 18, 307-311.
- WESTALL, R. F., PERKEY, M. N., & CHUTE, D. L. (1989). Millisecond timing on the Apple Macintosh: Updating Drexel's MilliTimer. *Behavior Research Methods, Instruments, & Computers*, 21, 540-547.

NOTES

1. A microphone can act as a response-triggering mechanism alone without a VAR when it is combined with a device called a pitch tracker or pitch follower. A pitch tracker takes an analog input signal and outputs a MIDI signal that includes both pitch and amplitude information. Pitch trackers have a disadvantage, in that they generally take at least 10 msec to convert audio information into MIDI form.
2. MIDI uses an 8-bit byte data structure, but an additional bit of information is sent before and after each 8-bit byte to separate 1 byte from another. This results in 10 bits of information being sent with each byte of data.
3. In each example presented here, a value of 144 will be used for a Note On status byte, indicating that MIDI Channel 1 will be used.
4. A signal can also be terminated with a 3-byte Note Off message that conveys channel number (128-143), pitch (0-127), and release velocity (0-127). The Note Off message is most useful with a keyboard

that provides information about the speed of the release envelope (i.e., how quickly a note begins and ends).

5. The manual also recommends that for program stability the MIDI-BASIC command `LIBRARY CLOSE` or `LIBTERM` should generally be called at the end of a program to reset certain system parameters to default conditions. This reportedly prevents system crashes. I have written and run several programs on a Macintosh IIcx and have found that the in-

clusion of `LIBRARY CLOSE` often causes the system crashes that the command is reputed to avoid. Leaving out this command seems to create no problems for the sorts of applications described in this paper when run on either a Macintosh Plus or a IIcx.

6. Many sound generators have less than 127 available patches or programs. Specifying the number of a nonexistent program will result in no sound being produced.