Getting MIDI from a Sun

Peter S. Langston

Bellcore
Morristown, New Jersey

ABSTRACT

Computer music software research and development requires a serious software development environment; either a special purpose system specifically attuned to music software tasks or a suitably powerful general purpose system must be found and interfaced to sound synthesis equipment. The Unix operating system provides the tools necessary for software development, however most Unix implementations have no facilities for interconnection with sound synthesis equipment. This report describes several ways to connect sound synthesizers to Unix minicomputer workstations such as those sold by Sun Microsystems.

March 23, 1990

Getting MIDI from a Sun

Peter S. Langston

Bellcore
Morristown, New Jersey

Introduction

The cost of computer-controlled sound synthesis equipment has plummeted in the last decade, largely as a result of the proliferation of consumer music products. That proliferation has largely been the result of the establishment of a musical instrument digital interface standard, MIDI[1]. Dozens of microcomputer systems are now offered that interface to MIDI, but the software development environments available on these systems are primitive at best. On the other hand, an excellent software development environment is generally available on minicomputers equipped with the Unix® operating system. Unfortunately, there is little overlap between the MIDI-equipped and Unix-equipped computer systems. One solution to this problem would be the installation of Unix operating systems on MIDI equipped microcomputers, but that is a difficult task (for various reasons). A simpler solution is the interfacing of MIDI to existing minicomputer systems running the Unix operating system.

Our lab at Bellcore currently has over one hundred minicomputer workstations from half a dozen manufacturers with many different machine architectures represented. Over the last four years we have attached MIDI interfaces to Multibus machines (e.g. Sun–2), VMEbus machines (e.g. Sun–3), AT bus machines (e.g. Sun–386i or ‘‘RoadRunner’’), and built interfaces that will connect to a serial port on any machine using the Zilog 8530 serial communications controller chip. The Multibus, VMEbus, and ATbus interfaces all connect to the Roland MPU-401 (or MPU-401 equivalent) MIDI interface, letting the MPU-401 do the timing for the MIDI data. On the other hand, the serial interface connects directly to MIDI synthesizers, requiring the workstation to do the timing (either in the user programs or in a kernel device driver).

This report first describes the MIDI and MPU data formats; then it introduces the Multibus, VMEbus, and ATbus interfaces briefly; and finally, it describes the serial port interface in enough detail to allow others to replicate it.

MIDI Format

Virtually all sound synthesizer manufacturers now make their devices read and/or write MIDI format. Many other kinds of devices also use MIDI data to control their operation – mixers, echo units, light controllers, etc. MIDI data is a serial, real-time data stream consisting of a sequence of 8-bit bytes representing synthesizer ‘‘events’’ such as note-start, note-end, volume change, and parameter selection. The authoritative description of the MIDI format is ‘‘MIDI 1.0 Detailed Specification’’ [MIDI89] published by the International MIDI Association, which should be consulted for detailed information. Particular items of interest in the MIDI specification are:

• The MIDI interface is asynchronous and operates at 31.25±1% kilobaud with 1 start bit, 1 stop bit, and 8 data bits. (This is similar to RS-232-C[2] at 31.25 Kbaud.)

• The MIDI interface is 5 mA current loop; each output can drive only one input. (This is not similar to RS-232-C which is voltage sensitive and can drive several inputs with one output.)

---

[1] ‘‘MIDI’’ is an acronym for ‘‘Musical Instrument Digital Interface.’’

[2] RS-232-C is an Electronic Industries Association (EIA) standard for ‘‘Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange’’ [EIA69].

• MIDI messages consist of one *status* byte followed by zero, one, or two data bytes, except in the case of messages that start with a *begin-system-exclusive* status byte, which may have an arbitrary number of data bytes followed by an *end-of-exclusive* status byte. The defined values and names for status bytes are shown in Figure 1.

MIDI COMMAND FORMATS		Status byte (hex)	Meaning	Number of data bytes
Channel (n = 0–F) Message		8n	Note Off	2
		9n	Note On	2
		An	Polyphonic After Touch	2
		Bn	Control Change	2
		Cn	Program Change	1
		Dn	Channel After Touch	1
		En	Pitch Bend	2
System Message	Excl.	F0	System Exclusive	?
	Common	F1	undefined	–
		F2	Song Position Pointer	2
		F3	Song Select	1
		F4	undefined	–
		F5	undefined	–
		F6	Tune Request	0
		F7	End of Sys.Excl.	0
	Real Time	F8	Timing Clock	0
		F9	undefined	0
		FA	Start	0
		FB	Continue	0
		FC	Stop	0
		FD	undefined	0
		FE	Active Sensing	0
		FF	System Reset	0

Figure 1 — MIDI Status Bytes

• MIDI describes ‘‘events’’ (e.g. hitting a key or moving a control) rather than ‘‘notes’’. In order to encode a ‘‘note’’ a pair of ‘‘events’ must be specified – a ‘‘key-on’’ event and a ‘‘key-off’’ event.

Note that nothing in the MIDI specification tells ‘‘when’’ an event is to occur; everything is ‘‘right now.’’

MPU Format

The MPU format gets its name from a hardware interface manufactured by the Roland Corporation called the MPU-401 that was one of the earliest interfaces available for interconnecting computers and synthesizers; several companies make similar interfaces that implement the same protocols in order to take advantage of existing software. (There are versions of the MPU-401 for the IBM PC’s, Apple computers, and for the Commodore 64, but none for Multibus or VMEbus computers.)

The MIDI standard defines key-on and key-off events as having two associated data bytes: a key-number (or pitch) and a velocity (or loudness). Key-numbers run from 0 to 127 (from C in octave -1 to G in octave 9, where middle C is called ‘‘C4’’ and ‘‘B3’’ is one half-step below middle C). Velocities run from 1 to 127 in arbitrary units with 1 being the slowest (quietest). The Roland MPU-401, when run in its ‘‘intelligent’’ mode, accepts ‘‘time-tagged’’ MIDI data, buffers it up, and spits it out at the times specified by the time-tags. The time-tags are relative delays indicating how many 120ths of a quarter-note to wait before sending out the next MIDI data. Thus a mezzo-forte quarter-note of middle C immediately followed by a forte sixteenth-note of the G above it could be encoded as shown in Figure 2.

Notice that timing resolution is roughly one 480th note (an MPU limitation) or 1 millisecond (a MIDI limitation); velocity resolution is 1 in 2^7 (a MIDI limitation), and pitch resolution is a semi-tone (a MIDI limitation)[3].

[3] It should be clear from this that MIDI was designed with keyboard instruments in mind.

time	status	key	vel	
0	90	3c	40	delay 0, key-on, key $3c_{16}=60_{10}$="C3", velocity $40_{16}=64_{10}$
78	80	3c	0	delay $78_{16}=120_{10}$=quarter note; key-off, key $3c_{16}=60_{10}$="C3"
0	90	43	54	delay 0, key-on, key $43_{16}=67_{10}$="G3", velocity $54_{16}=84_{10}$
1e	80	43	0	delay $1e_{16}=30_{10}$=sixteenth note; key-off, key $43_{16}=67_{10}$="G3"

Figure 2 — Time-Tagged MIDI Example

Multibus and VMEbus Interfaces

These interfaces are relatively simple devices that connect the MPU-401 to the appropriate bus lines. Appendix 1 contains a schematic for one version of the Multibus interface. The paper "MIDI Music Software for Unix" [Hawley85] describes this interface in more detail.

In order to use the MPU-401 effectively a kernel device driver and character special file entry (typically "/dev/mpu0") are required. The Multibus interface board supports a single MPU-401, whereas the VMEbus interface board supports four MPU-401s ("/dev/mpu0" through "/dev/mpu3"). The MPU device driver performs many complex functions; its design is beyond the scope of this paper. The Unix MPU device driver is available from several sources (contact the author for further information).

Both of these interfaces have been used successfully in demanding applications. The Multibus interface was used for three years in an unattended system that gave music demos to people calling in from telephones all over the world [LANGST86], handling as many as 400 demos a day. Last year the Multibus host computer (with Multibus interface) was replaced with a VMEbus system (with the VMEbus version of the interface) that now gives the demos.

ATbus Interface

The Sun RoadRunner workstation (also known as the Sun 386i) is one of the easiest of the Unix workstation products to interface to MIDI. It contains several empty slots for peripherals that are designed for the bus used in the IBM PC series. Roland makes a special version of the MPU-401 called the MPU-IPC which plugs directly into this bus (and is cheaper than the MPU-401). The only drawback is that the MPU-IPC comes wired for interrupt level 2 and that level is already in use on the RoadRunner. It is necessary to cut a trace on the interface board of the MPU-IPC and insert a jumper to configure it for interrupt level 4. Figure 3 shows the wiring and bit positions for the two jumper areas on the MPU-IPC interface card (unla-

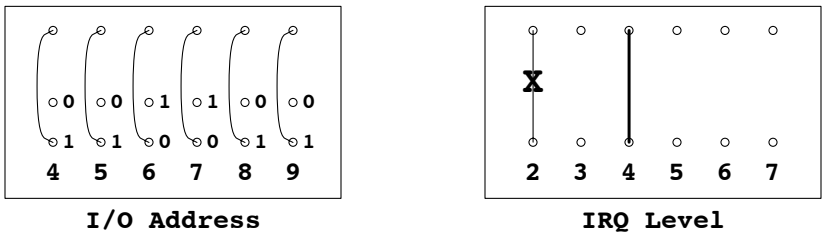


Figure 3 — MPU-IPC Jumpers

beled on the card). The trace at IRQ level 2 should be cut at the "X" and a jumper should be installed at IRQ 4 (marked with a heavy line). Note that the factory default for the I/O address is 0x330 (bits 11, 10, 3, 2, 1, and 0 have no jumpers and are fixed at 0, so the bits are 001100110000 from bit 11 through bit 0). Beware: the locations of the 1's and 0's in the I/O address jumpers stagger strangely.

Use of the MPU-IPC connected to the RoadRunner's AT bus also requires the use of the MPU device driver (as mentioned above).

Serial Port Interface

The serial port interface came about out of the need to get MIDI out of a ‘‘desktop’’ workstation that had no extra card slots to accomodate an interface card. The similarities (and dissimilarities) between MIDI and RS-232-C communications have already been mentioned; there are just two major differences to be overcome in using an RS-232-C port to send and receive MIDI: (1) speed – MIDI requires 31.25 Kbaud ±1%, i.e. 30.9 to 34.4 Kbaud, but the nearest standard RS-232-C value is 19.2 Kbaud (although many devices offer 38.4 Kbaud which is higher than the EIA specification’s ‘‘...nominal upper limit of 20,000 bits per second.’’); (2) electrical characteristics – RS-232-C is a voltage based protocol; ONE and ZERO are defined as voltage ranges relative to ground. MIDI is a current based protocol; ONE and ZERO are defined as loop current values.

Many Unix-based workstations use the Zilog 8530 Serial Communications Controller chip for their serial ports. To set the baud rate on this chip the system must provide a divisor whose effect on the baud rate is:

$$Baudrate = \frac{153,600}{Divisor + 2}$$

From this equation it should be apparent that a Divisor of 2 will provide 38.4 Kbaud and a Divisor of 6 will provide 19.2 Kbaud, both fairly common data rates. It should also be apparent that a Divisor of 3 will provide 30.7 Kbaud. This is 1.7% below the desired value of 31.25 Kbaud and 0.7% below the specified range for MIDI, but we decided to try it anyway. We used a standard debugging program to alter the divisors table (zs_speeds) in the UNIX serial device driver, replacing the value for 200 baud (766) with 3. It has worked perfectly with every piece of MIDI equipment we have tried (more than a dozen so far). This is not surprising since the MIDI protocol is asynchronous and must resynchronize on each message; after ten bits the cumulative error is still less than one fifth of a bit time.

The remaining problem was to convert the RS-232-C voltage levels to 5 milliampere current loop. Many products are available to convert RS-232-C to *20 milliampere* current loop, but since we not only needed to write current loop data, but also needed to read *5 milliampere* current loop data, we needed a converter with greater sensitivity. To this end we designed the converter in Figure 4. Note that we also built a power supply, using a Radio Shack 9v ‘‘power supply’’ as our source of unregulated power. We could not use the power from the RS-232-C interface because the RS-232-C specification limits the current available to less than the 5 mA needed for our current loop interface. (Further, we wanted to be able to use the converter in situations where the RS-232-C side is passive.) The complete parts list for construction of the device is shown in Figure 5.

#	Capacitors	#	ICs, Semiconductors	#	Resistors & Misc.
1	0.01 mfd	1	2300 optoisolator	1	100 ohm ¼ watt
2	0.1 mfd	1	6N138 optoisolator	5	226 ohm ¼ watt
1	0.47 mfd 10V	1	6N136 optoisolator	1	2.7K ohm ¼ watt
1	100 mfd 20V	1	78L05 voltage regulator	1	DB25S connector
1	150 mfd 6V	1	NCN0512 voltage converter	2	Switchcraft 57GB5F jack
		2	misc switching diodes		misc hdwe, box, etc.

Figure 5 — Parts for the RS232/MIDI Converter

Timing With the Serial Port Interface

Since the serial port interface reads and writes MIDI directly without the intervention of a buffering device like the MPU-401 it becomes the workstation’s job to send the MIDI data at just the right time. Many workstations only provide 20 millisecond resolution in their timers; this turns out to be only a minor problem, however. A fiftieth of a second is much smaller than the variation typically introduced by human musicians. Further, the MIDI protocol takes 1/3 of a millisecond to send each byte, so a simple key-on command takes a millisecond; It is quite common to require a dozen notes to start ‘‘simultaneously’’ and not notice the resulting spread of 12 milliseconds in the output.

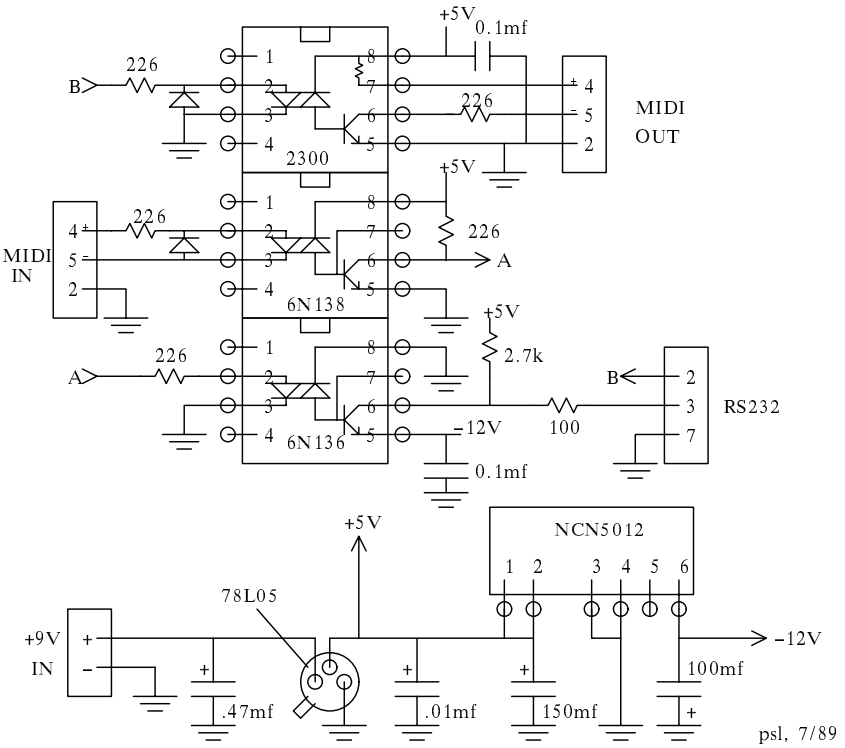


Figure 4 — RS-232-C to MIDI Converter

What does pose a problem is the effect of being swapped out while playing MIDI data and not being able to send data for several tenths of a second. Raising the priority of the playing process helps curtail this problem, but does not solve it. The only realistic solution is to put the timing function in the kernel. Contact the author for details on the availability of a MIDI/serial device driver.

Summary

Four methods of connecting MIDI peripherals to Unix-based workstations have been presented; one each for Multibus, VMEbus, and AT bus workstations, and one for workstations that use the Zilog 8530 Serial Communications Controller chip. These interfaces have provided crucial support for several projects over the last four years [LANGST89][LANGST90a][LANGST90b]. The interfaces all work.

Acknowledgements

Gareth Loy of UCSD and Michael Hawley of Next Computers (at Lucasfilm at the time) provided the initial version of the Multibus interface [HAWLEY85]. Dave Cumming of M.I.T. Media Lab assembled the VMEbus interface. Daniel Steinberg of Sun Microsystems collaborated on improving the MPU kernel driver. Namon Jackson of Bellcore provided electronic engineering expertise of a far higher caliber than should have been required for the design of a simple optoisolator circuit. Finally, Doug Kingston of Morgan Stanley pointed out that sticking a 3 in the Unix kernel's *zs_speeds* array could make a serial port data rate very close to the MIDI rate.

References

EIA69 Electronic Industries Association, 1969. "EIA Standard RS-232-C Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange."

HAWLEY86 Hawley, M. 1986. ''MIDI Music Software for Unix.'' *Proceedings of the Usenix Summer '86 Conference*.

LANGST86 Langston, P.S. 1986. ''(201) 644-2332 • Eedie & Eddie on the Wire, An Experiment in Music Generation.'' *Proceedings of the Usenix Summer '86 Conference*.

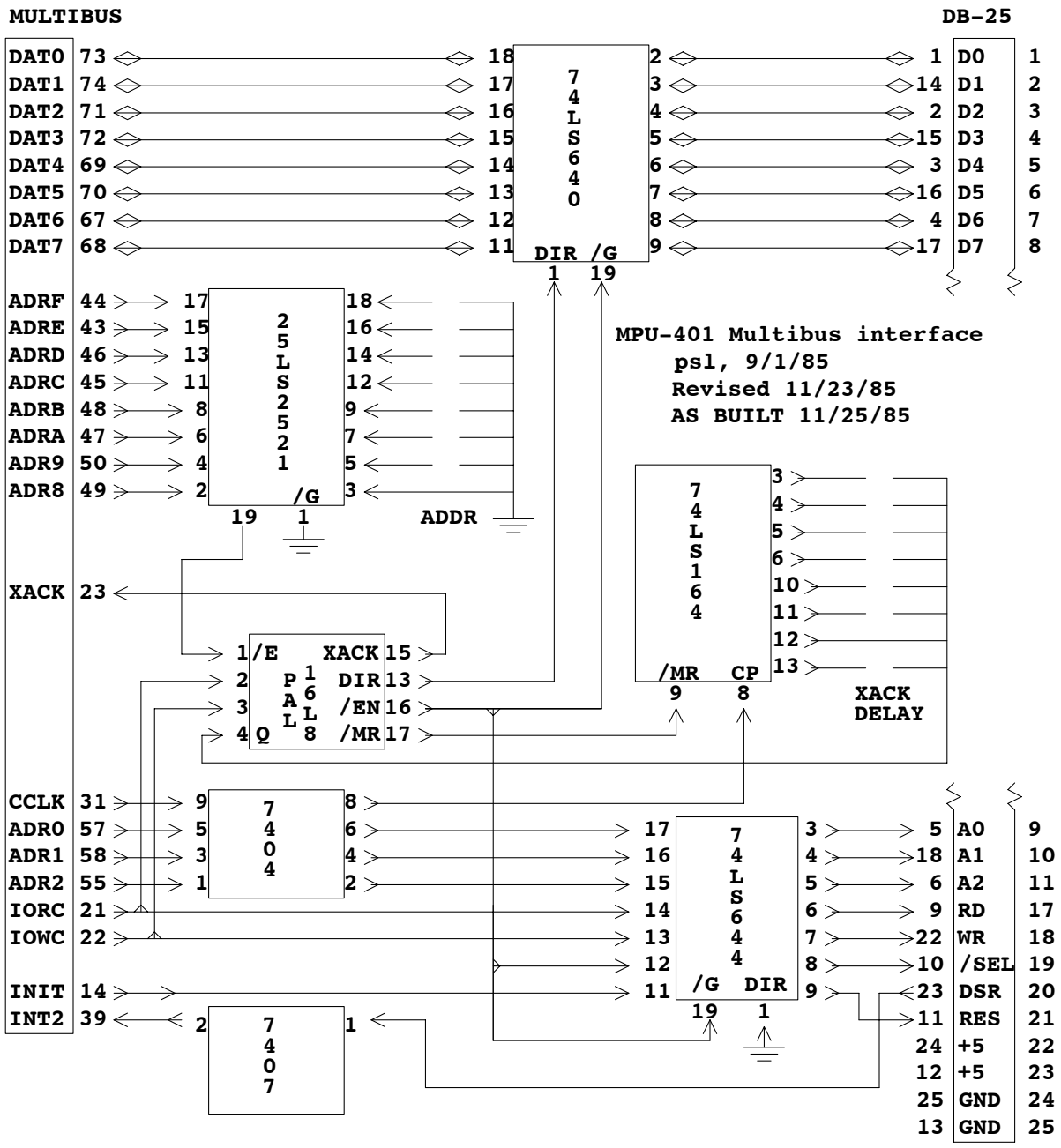
LANGST89 Langston, P.S. 1989. ''Six Techniques for Algorithmic Music Composition.'' Bellcore Technical Memorandum #ARH-013020.

LANGST90a Langston, P.S. 1990. ''Little Languages for Music.'' to appear in *Computing Systems*.

LANGST90b Langston, P.S. 1990. ''PellScore – An Incidental Music Generator'' Bellcore Technical Memorandum #ARH-016281.

MIDI89 The International MIDI Association, 1989. *MIDI 1.0 Detailed Specification, Document version 4.1*. I.M. A., 5316 W. 57th St., Los Angeles, CA 90056.

APPENDIX


MULTIBUS
DB-25
DAT0 73
DAT1 74
DAT2 71
DAT3 72
DAT4 69
DAT5 70
DAT6 67
DAT7 68
74LS640
DIR /G
1 19
D0 1
D1 2
D2 3
D3 4
D4 5
D5 6
D6 7
D7 8
ADRF 44
ADRE 43
ADRD 46
ADRC 45
ADRB 48
ADRA 47
ADR9 50
ADR8 49
25LS2521
/G
ADDR
MPU-401 Multibus interface
psl, 9/1/85
Revised 11/23/85
AS BUILT 11/25/85
74LS164
/MR CP
XACK DELAY
XACK 23
PAL16L8
1/E
2
3
4 Q
XACK 15
DIR 13
/EN 16
/MR 17
CCLK 31
ADR0 57
ADR1 58
ADR2 55
IORC 21
IOWC 22
INIT 14
INT2 39
7404
7407
74LS644
/G DIR
A0 9
A1 10
A2 11
RD 17
WR 18
/SEL 19
DSR 20
RES 21
+5 22
+5 23
GND 24
GND 25