BEAT-IT: A Drum Sensor Interface for the Atari ST

If you have an Atari ST and know how to solder, here's what you need to start pounding out MIDI program numbers that'll turn your synth into a drum synthesizer.

By David Snow

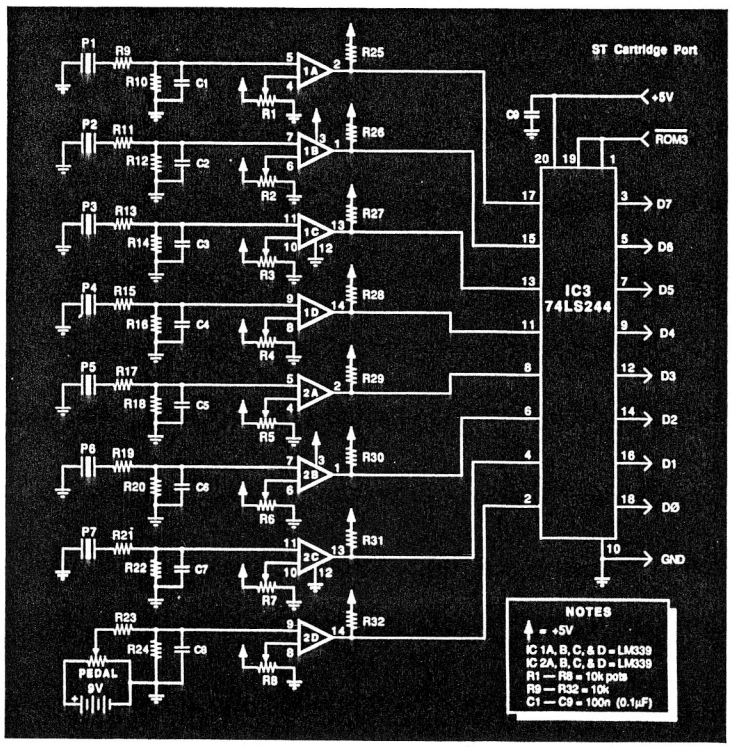

FIG. 1: Beat-It drum interface schematic.

You might think that percussionists are a well-adjusted lot, given that they regularly vent their aggression for fun and profit, and they are; it's just that some of them look and act like Animal from *The Muppet Show* (then again, he was a good drummer, wasn't he?). Hey, I'm not criticizing; I *want* to be like that. As a matter of fact, one can get pretty tired of being pigeonholed as some bow-tied, ivory-tower, pseudo-intellectual, button-pushing nerd. I want to rock. I want to roll. I want to get down, crank it up, kick out the jams, bite off chicken heads, and . . . you know what I mean. The problem is that it's just not spiritually moving to spend hours at a stretch entering step-time data into your sequencer with tiny, plastic buttons. You've got to keep in touch with the Big Reality, play your music in the Here and Now. For *that* transcendental purpose, nothing, you might say, beats a drum.

Since nobody's gonna let you beat up on the furniture, you have to get an instrument. Of course, money to indulge your muse is no object unless you run into a snag ("Hey Pop, how about 15,000 weeks advance on my allowance?"). Well, if the Real Reality says you need something substantial, but that Hot Rockers Drum Kit on sale at the local toy outlet isn't going to make the grade, don't panic.

You have a CZ-101 (or other multi-timbral synth)? You have an ST? You have a soldering iron? You're covered. Beat-It is a drum sensor interface that plugs into the ST's cartridge slot and turns that synth into a drum synth with eight, count 'em, eight different sounds available at your fingertips. The sensors (triggers) can be piezo elements, mics, pedals, all kinds of junk. The possibilities are mind-boggling, and the boggling follows forthwith.

THE CIRCUIT

The interface circuitry (**Fig. 1**) is simple enough. Quad comparators IC1 and IC2 detect voltage inputs from the sensors. If the input exceeds the threshold set by trimmers R1 to R8, the comparator output goes high. Octal buffer IC3 has tri-state outputs that isolate the comparator outputs from the computer's data bus unless enabled by strobing pins 1 and 19 low (see also **Fig. 2**). The ST's cartridge port provides two decoding lines: ROM3, which goes low when reading from addresses $FA0000 to $FAFFFF, and ROM4, which goes low when reading from $FB0000 to $FBFFFF. By reading address $FA0000, one enables the octal buffer and loads the comparator states onto data lines D0 to D7.

The hardest part about building the circuit is obtaining a circuit card to mate with the cartridge port (**Fig. 3**), as the contacts use an unusual 0.079-inch spacing. I

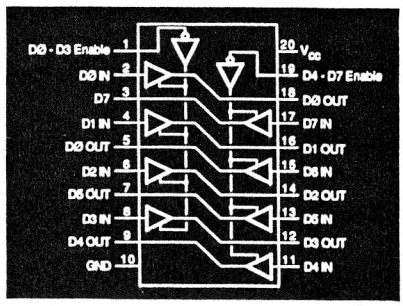

FIG. 2: 74LS244 Octal Tri-State™ Buffer pinout.

BEAT-IT
Drum Sensor Interface for the Atari ST

```
defwrd a-z

dim static prog_array(8)
dim static note_array(8)
dim static chan_assign(4)

' INITIALIZE PROGRAM/NOTE ASSIGNMENT
for drum=0 to 7
     prog_array(drum)=drum+32: note_array(drum)=60
next

display:
     clearw 2
     print "DRUM:     PROGRAM:  NOTE:"
     for drum=0 to 7
          print drum+1;tab(10);prog_array(drum);tab(20);note_array(drum)
     next
     print: print " ENTER DRUM NUMBER, OR PRESS 'RETURN' TO START:";

get_input:
     input " ", drum$
     if drum$="" then
          print
          print " READY TO PLAY. PRESS ANY KEY TO STOP (CTRL-C TO EXIT)."
          gosub play
          goto display
     endif

     drum=val(drum$)-1
          if drum <0 or drum>7 then display

     input " ENTER NEW PROGRAM NUMBER OR PRESS 'RETURN': ", prog$
          if prog$="" then get_note
          prog_array(drum)=val(prog$)

     get_note:
          input " ENTER NEW NOTE VALUE OR PRESS 'RETURN': ", note$
          if note$="" then display
          note_array(drum)=val(note$)
     goto display

play:
     out 3,176: out 3,126: out 3,4: '   MONO MODE ON

     for channel=0 to 3:'     ASSIGN PROGRAMS TO CHANNELS 1-4
          chan_assign(channel)=channel
          out 3,192+channel
          out 3,prog_array(channel)
     next

     new_channel=0: old_input=0

     while inp(-2)<>-1
          drum_input=peek_w(&hFA0000)-&hFF00
          if drum_input<>0 and drum_input<>old_input then
               for drum=0 to 7
                    test=drum_input and 2^drum
                    if test<>0 then
                         channel=0
                         find_channel:
                              if chan_assign(channel)=drum then note_on
                              channel=channel+1
                              if channel<4 then find_channel

                         chan_assign(new_channel)=drum
                         channel=new_channel
                         new_channel=(new_channel+1) mod 4

                         out 3,192+channel
                         out 3,prog_array(drum)
```

```
                        note_on:
                              out 3,144+channel
                              out 3,note_array(drum)
                              out 3,64
                              out 3,note_array(drum)
                              out 3,0
                        endif
                  next drum
            endif
            old_input=drum_input
      wend
      junk=inp(2)
return
```

LISTING 1: Beat-It software listing.

kluged my own connector using the Radio Shack PC board kit, drawing the card-edge fingers on the copper-clad side of the board with a resist-ink pen, and putting Bishop Graphics EZ Circuit adhesive copper traces on the other side. I built the interface circuit on a separate prototype board (Radio Shack #276-154) and connected it to the card-edge board with a 16-pin DIP jumper. If all that is too much trouble, a custom wire-wrap plugboard for the ST cartridge port is available from Douglas Electronics (see Parts List). Circuit construction is not too critical, but make wire runs short and direct and keep the comparator output leads away from the input leads to prevent oscillation.

For drum pads, I used wooden plaques purchased from a crafts shop and attached piezo-element sensors (Radio Shack #273-073) with epoxy (see "Quad Piezo-Electric Drum Trigger" by Chris Lucht, July '86 **EM**). The plaques are cushioned on a bed of styrofoam for isolation and mounted on a wooden frame at a comfortable angle for playing. For a bass pedal, I use a volume pedal reconfigured as a "voltage pedal" (**Fig. 4**). When using a pedal as the input sensor, the associated threshold trimmer should be adjusted so that the comparator turns on when the pedal is almost completely depressed, which gives the best "feel." That's all there is to Beat-It hardware.

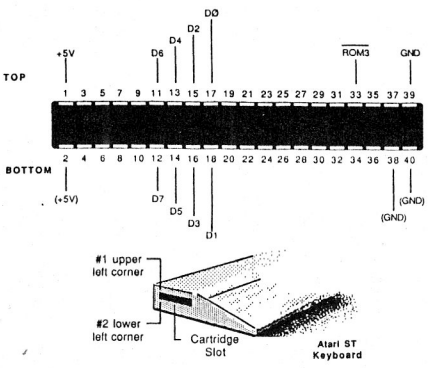

FIG. 3: Atari ST cartridge port (viewed looking into computer).

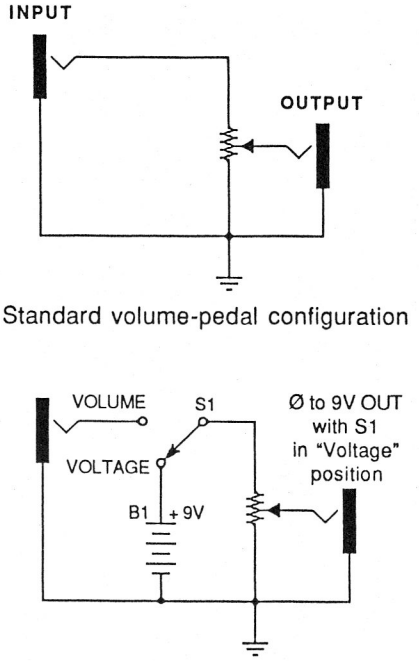

FIG. 4: Converting a volume pedal into a drum footpedal.

THE SOFTWARE

The main function of the controlling software is to input data from the comparators, test each comparator's bit, and output a MIDI message if it's high. The only special software trick Beat-It performs is to make it seem like you have eight different voices available when, in the case of the CZ-101, you really only have four (using two different timbres per note in a TX81Z Performance also yields a total of four voices). Although the multi-timbral CZ is limited to playing up to four different timbres simultaneously, each triggered by information on a different MIDI channel, you can instantly change an active

voice to another sound by sending a MIDI Program Change message over a designated MIDI channel.

A brute force software solution would be to send a MIDI Program Change message before every Note On message, but this isn't satisfactory because it slows down the synthesizer's response. The best approach is to let the Atari sort out the details and send Program Change messages only when necessary, that is, whenever a sound program number is requested that isn't already assigned to one of the four synthesizer channels.

In order to do this, the software maintains a list of four of the eight sensors (which are numbered from 0 to 7) whose programs are currently assigned. If the computer detects a sensor whose number is on the list, it outputs that sensor's MIDI Note On over the channel corresponding to its place on the list. If the sensor number is not on the list, it is placed on the list in a position determined by a rotating pointer. Then, its MIDI Program Change is sent to the synth on the channel determined by its place on the list, followed by the Note On information (see sidebar).

The software program (**Listing 1**) was written using the *LDW BASIC* compiler, which produces fast .PRG files that do not require an interpreter. Anyone who wants to write useful software for the ST, but doesn't want to learn C, should get a BASIC compiler. Beat-It doesn't exploit the elegance of GEM (although LDW BASIC provides full access to it), but that's the price of simplicity. It gets the job done.

Enough technical stuff, now let's BEAT IT—hard.

CUSTOMIZING

The only user input Beat-It requires is MIDI program and note values for each of the sensors. The note range is from 36 (lowest C) to 84 (highest C). On the CZ-101, the programs are assigned as follows:

Presets	program numbers 0 to 15
Internal	program numbers 32 to 47
Cartridge	program numbers 64 to 79

The software's default settings assign the first eight internal programs to the eight sensors, each with a MIDI note number of 60 (middle C).

After typing in and saving the program, run Beat-It and enter the MIDI program and note values of your choice for each sensor. Press the Return key to activate the interface, and adjust trimpots R1 to R7 so that a strike on the drum pads produces a single note with no multiple (false) triggering.

The best patches to use are percussive sounds with no sustain; a selection of percussion patches accompanies my "Drumbox" article in the February '88 issue of

NOTES ON MIDI

MIDI Note On and Note Off messages consist of three, 8-bit bytes of information indicating type of command (first half of first byte), channel number (nnnn), note number, and key velocity.

Note Off:	1000nnnn	note####	velocity
Note On:	1001nnnn	note####	velocity

Program Change messages are two bytes long and include channel number and program number.

Program Change:	1100nnnn	program#

PARTS LIST

RESISTORS (¼W, 5% tolerance)

R1-R8	10k trimpot
R9-R32	10k

CAPACITORS

C1-C9	100n (0.1 μF) ceramic disc

SENSORS

P1-P7	piezo element (Radio Shack #273-073)

SEMICONDUCTORS

IC1, IC2	LM339 quad comparator
IC3	74LS244 octal buffer

MISCELLANEOUS

Circuit card: custom wire-wrap plugboard for Atari ST from Douglas Electronics, 718 Marina Blvd., San Leandro, CA 94577 ($10; catalog #33-DE-40) or equivalent (see text).

EM (pages 39-43).

If you really want to get strange, you can use other kinds of input devices for sensors. Any device or circuit that outputs up to 10V will trigger the comparators, such as mechanical and electronic switches (how about mercury-switch "maracas"?), light-sensitive devices incorporating photocells, pressure-sensitive voltage dividers made with conductive foam (see "Build an Electric Drum Pad" by Thomas Henry, December '84 *Polyphony*), analog envelope followers, or line-level audio. I've used Craig Anderton's Envelope Trigger device (*Keyboard*, September '83) to trigger Beat-It from a microphone. What I'd really like to see is an infrared switching array that triggers the comparators when the performer interrupts the beams with his hands: invisible drums!

What a concept.

David Snow *took up the trumpet at age 11 but turned to MIDI in his twilight years to compensate for instrumental technique he couldn't acquire honestly. He holds degrees in composition from Eastman and Yale and is the recipient of numerous grants, awards, and commissions, including two Composer Fellowships from the National Endowment for the Arts.*

By David Snow

ST BASIC TRANSLATION OF "BEAT-IT"

If you've been honing your computer programming chops on the Atari ST, you've probably come up against the frustrating quirks and limitations of ST BASIC. Many published programs are written in more efficient languages, which is great as long as you have the required compiler or interpreter. Unfortunately, many novices lack that luxury, but it is sometimes possible to translate from one language to another without an unacceptable compromise in performance.

A reader recently requested an ST BASIC translation of the software for "Beat-It," the drum sensor interface project that appeared in the December 1988 issue of **EM**. Because ST BASIC is too sluggish for time-critical tasks, the program incorporates a machine code module that handles all interface and MIDI business. This version works in the same manner as the LDW BASIC version; refer to the original article for more information. Simply type in **Listing 1** (use an ASCII editor, such as 1st Word with WP mode off; the ST BASIC editor is a pain), save the file, load BASIC, and run the program.

ST BASIC Version of "Beat It"

LISTING 1

```
10  'BEAT IT - Drum Sensor Interface for the  Atari ST
20  'by David Snow; ST BASIC version 1/29/89
30  '
40  defint a-z
50  dim  play(149)
60  play(0)=0:          play#=varptr(play(0))
70  progarray(0)=0:     progarray#=varptr(progarray(0))
80  notearray(0)=0:     notearray#=varptr(notearray(0))
90  '
100 def seg=play#
110 for index#=0 to 296
120 read byte$: poke index#,val("&h"+byte$)
130 next
140 data 60,1A,00,00,01,0C,00,00,00,00,00,00,00,00,00,00
150 data 00,00,00,00,00,00,00,00,00,00,FF,FF,48,E7,FF,FE
160 data 20,6F,00,42,2C,50,2A,68,00,04,49,FA,00,F8,30,3C
170 data 00,B0,61,00,00,DC,30,3C,00,7E,61,00,00,D4,30,3C
180 data 00,04,61,00,00,CC,3E,3C,00,03,3C,3C,00,06,19,87
190 data 70,00,30,3C,00,C0,D0,47,61,00,00,B6,30,36,60,00
200 data 61,00,00,AE,55,46,51,CF,FF,E6,42,47,42,46,3F,3C
210 data 00,02,3F,3C,00,01,4E,4D,58,8F,4A,40,66,00,00,8C
220 data 3A,39,00,FA,00,00,02,45,00,FF,4A,45,67,00,00,76
230 data BA,46,67,00,00,70,38,3C,00,07,34,04,E3,4A,09,05
240 data 67,00,00,5E,36,3C,00,03,B8,34,30,00,67,00,00,2A
250 data 51,CB,FF,F6,19,84,70,00,36,07,52,47,0C,47,00,04
260 data 66,00,00,04,42,47,30,3C,00,C0,D0,43,61,00,00,42
270 data 30,36,20,00,61,00,00,3A,30,3C,00,90,D0,43,61,00
280 data 00,30,30,35,20,00,61,00,00,28,30,3C,00,40,61,00
290 data 00,20,30,35,20,00,61,00,00,18,42,40,61,00,00,12
300 data 51,CC,FF,98,3C,05,60,00,FF,66,4C,DF,7F,FF,4E,75
310 data 3F,02,3F,00,3F,3C,00,03,3F,3C,00,03,4E,4D,5C,8F
320 data 34,1F,4E,75,00,00,00,03,00
330 ' INITIALIZE PROGRAM/NOTE ASSIGNMENT
340 for drum=0 to 7
350 progarray(drum)=drum+32: notearray(drum)=60
360 next
370 fullw 2
380 '
390 display:
400 clearw 2: gotoxy 0,0
410 print "DRUM:     PROGRAM:  NOTE:"
420 for drum=0 to 7
430 print drum+1;tab(10);progarray(drum);tab(20);notearray(drum)
440 next
450 print: print " ENTER DRUM NUMBER TO EDIT, OR PRESS 'RETURN' TO START:";
460 '
470 getinput:
480 input " ", drum$
490 if len(drum$)<>0 then goto getdata
500 print
510 print " READY TO PLAY. PRESS ANY DRUM NUMBER TO STOP (CTRL-C TO EXIT)."
520 call play#(progarray#,notearray#)
540 goto display
550 '
560 getdata:
570 drum=val(drum$)-1
580 if drum <0 or drum>7 then goto display
590 input " ENTER NEW PROGRAM NUMBER OR PRESS 'RETURN': ", prog$
600 if len(prog$)=0 then goto getnote
610 progarray(drum)=val(prog$)
620 '
630 getnote:
640 input " ENTER NEW NOTE VALUE OR PRESS 'RETURN': ", note$
650 if len(note$)=0 then goto display
660 notearray(drum)=val(note$)
670 goto display
```